Librerie LASD 2024/2025

Generated by Doxygen 1.13.2

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 lasd Namespace Reference

Classes

- class ClearableContainer
- class Container
- class DictionaryContainer

  La classe DictionaryContainer definisce al suo interno tutti quei metodi che permettono ad una struttura dati di funzionare come un dizionario.
- class LinearContainer

  Classe astratta che rappresenta un contenitore lineare accessibile per posizione.
- class List
- class MappableContainer

  Classe astratta che estende TraversableContainer, permettendo la modifica degli elementi tramite funzioni mappanti.
- class MutableLinearContainer
- class OrderedDictionaryContainer

  Classe che estende DictionaryContainer con capacità di ordinamento.
- class PostOrderMappableContainer

  Estensione di MappableContainer che specifica l'ordine PostOrder per la mappatura.
- class PostOrderTraversableContainer

  Classe astratta per contenitori con traversata in post-ordine.
- class PreOrderMappableContainer

  Estensione di MappableContainer che specifica l'ordine PreOrder per la mappatura.
- class PreOrderTraversableContainer

  Classe astratta per contenitori con traversata in pre-ordine.
- class ResizableContainer
- class Set
- class SetLst
- class SetVec
- class SortableLinearContainer

  Classe astratta che rappresenta un contenitore lineare ordinabile.
- class SortableVector

  Classe concreta che estende Vector con capacità di ordinamento.
- class TestableContainer

  Classe astratta che estende Container con un metodo per il test di esistenza.
- class TraversableContainer

  Classe astratta di contenitore traversabile.
- class Vector

  Classe concreta che rappresenta un vettore dinamico di elementi.

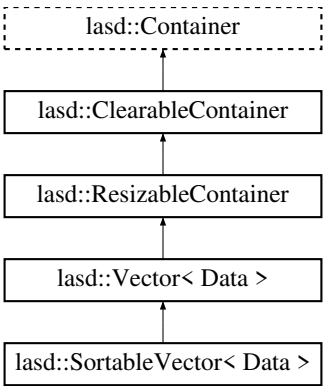
Chapter 6

Class Documentation

6.1 lasd::ClearableContainer Class Reference

`#include <container.hpp>`

Inheritance diagram for lasd::ClearableContainer:



Public Member Functions

- virtual ~ClearableContainer ()=default

  Destructor.
- ClearableContainer & operator= (const ClearableContainer &)=delete

  Copy assignment.
- ClearableContainer & operator= (ClearableContainer &&) noexcept=delete

  Move assignment.
- bool operator== (const ClearableContainer &Ccon) const noexcept=delete

  Comparison operators.
- bool operator!= (const ClearableContainer &Ccon) const noexcept=delete
- virtual void Clear ()=0

  Non avendo modo di sapere come svuotare una struttura dati generica, il metodo Clear è definito come un metodo virtuale puro, ovvero un metodo che deve essere implementato dalle classi che estendono la classe ClearableContainer.

Public Member Functions inherited from lasd::Container

- virtual ~Container ()=default

  Destructor.
- Container & operator= (const Container &)=delete

  Copy assignment of abstract types is not possible.
- Container & operator= (Container &&) noexcept=delete

  Move assignment of abstract types is not possible.
- bool operator== (const Container &) const noexcept=delete
- bool operator!= (const Container &) const noexcept=delete
- virtual bool Empty () const noexcept

  La funzione Empty() controlla se la struttura è vuota (concrete function should not throw exceptions)
- virtual ulong Size () const noexcept

  La funzione Size() restituisce il numero di elementi presenti nella struttura (concrete function should not throw exceptions)

Additional Inherited Members

Protected Member Functions inherited from lasd::Container

- Container ()=default

  Default constructor.

Protected Attributes inherited from lasd::Container

- ulong size = 0

6.1.1 Detailed Description

La classe ClearableContainer rappresenta una classe astratta per tutte quelle strutture dati che possono essere svuotate, ovvero che possono essere ripristinate allo stato iniziale,

Definition at line 48 of file container.hpp.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 ~ClearableContainer()

```
virtual lasd::ClearableContainer::~ClearableContainer ()  [virtual], [default]
```

Destructor.

6.1.3 Member Function Documentation

6.1.3.1 Clear()

```
virtual void lasd::ClearableContainer::Clear ()  [pure virtual]
```

Non avendo modo di sapere come svuotare una struttura dati generica, il metodo Clear è definito come un metodo virtuale puro, ovvero un metodo che deve essere implementato dalle classi che estendono la classe ClearableContainer.

Implemented in lasd::ResizableContainer, and lasd::Vector< Data >.

6.1.3.2 operator"!=()

```
bool lasd::ClearableContainer::operator!= (
            const ClearableContainer & Ccon) const  [delete], [noexcept]
```

6.1.3.3 operator=() [1/2]

```
ClearableContainer & lasd::ClearableContainer::operator= (
            ClearableContainer && )  [delete], [noexcept]
```

Move assignment.

6.1.3.4 operator=() [2/2]

```
ClearableContainer & lasd::ClearableContainer::operator= (
            const ClearableContainer & )  [delete]
```

Copy assignment.

6.1.3.5 operator==()

```
bool lasd::ClearableContainer::operator== (
            const ClearableContainer & Ccon) const  [delete], [noexcept]
```

Comparison operators.

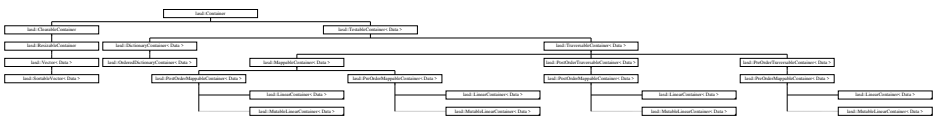
The documentation for this class was generated from the following file:

- Exercise1/container/container.hpp

6.2 lasd::Container Class Reference

```
#include <container.hpp>
```

Inheritance diagram for lasd::Container:



Public Member Functions

- virtual ~Container ()=default

  Destructor.
- Container & operator= (const Container &)=delete

  Copy assignment of abstract types is not possible.
- Container & operator= (Container &&) noexcept=delete

  Move assignment of abstract types is not possible.
- bool operator== (const Container &) const noexcept=delete
- bool operator!= (const Container &) const noexcept=delete
- virtual bool Empty () const noexcept

  La funzione Empty() controlla se la struttura è vuota (concrete function should not throw exceptions)
- virtual ulong Size () const noexcept

  La funzione Size() restituisce il numero di elementi presenti nella struttura (concrete function should not throw exceptions)

Protected Member Functions

- Container ()=default

  Default constructor.

Protected Attributes

- ulong size = 0

6.2.1 Detailed Description

La classe Container fornisce la prima classe a partire dalla quale andremo a definire le nostre interfacce. È l'analogo della classe Object in Java.

Definition at line 11 of file container.hpp.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 Container()

```
lasd::Container::Container ()  [protected], [default]
```

Default constructor.

6.2.2.2 ~Container()

```
virtual lasd::Container::~Container ()  [virtual], [default]
```

Destructor.

6.2.3 Member Function Documentation

6.2.3.1 Empty()

```
virtual bool lasd::Container::Empty () const  [inline], [virtual], [noexcept]
```

La funzione Empty() controlla se la struttura è vuota (concrete function should not throw exceptions)

Returns

true se la struttura è vuota, false altrimenti

Definition at line 36 of file container.hpp.

6.2.3.2 operator"!=()

```
bool lasd::Container::operator!= (
            const Container & ) const  [delete], [noexcept]
```

6.2.3.3 operator=() [1/2]

```
Container & lasd::Container::operator= (
            const Container & )  [delete]
```

Copy assignment of abstract types is not possible.

6.2.3.4 operator=() [2/2]

```
Container & lasd::Container::operator= (
            Container && )  [delete], [noexcept]
```

Move assignment of abstract types is not possible.

6.2.3.5 operator==()

```
bool lasd::Container::operator== (
            const Container & ) const  [delete], [noexcept]
```

6.2.3.6 Size()

```
virtual ulong lasd::Container::Size () const  [inline], [virtual], [noexcept]
```

La funzione Size() restituisce il numero di elementi presenti nella struttura (concrete function should not throw exceptions)

Returns

Il numero di elementi nella struttura

Definition at line 42 of file container.hpp.

6.2.4 Member Data Documentation

6.2.4.1 size

```
ulong lasd::Container::size = 0  [protected]
```

L'attributo size indica il numero di elementi presenti nel Container

Definition at line 15 of file container.hpp.

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

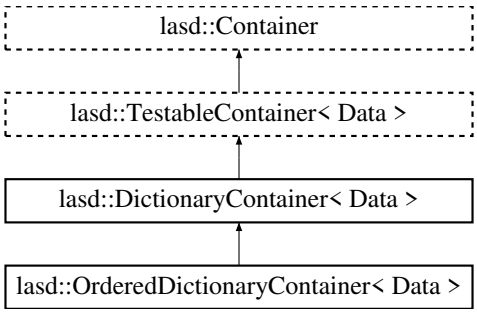
- Exercise1/container/container.hpp

6.3 lasd::DictionaryContainer< Data > Class Template Reference

La classe DictionaryContainer definisce al suo interno tutti quei metodi che permettono ad una struttura dati di funzionare come un dizionario.

```
#include <dictionary.hpp>
```

Inheritance diagram for lasd::DictionaryContainer< Data >:



Public Member Functions

- virtual ~DictionaryContainer ()=default

  Destructor.
- DictionaryContainer & operator= (const DictionaryContainer &)=delete

  Copy assignment.
- DictionaryContainer & operator= (DictionaryContainer &&)=delete

  Move assignment.
- bool operator== (const DictionaryContainer &) const noexcept=delete
- bool operator!= (const DictionaryContainer &) const noexcept=delete
- virtual bool Insert (const Data &val)=0

  Inserisce un elemento nella struttura dati.
- virtual bool Insert (Data &&val)=0

  Inserisce un elemento usando move semantics.
- virtual bool Remove (const Data &val)=0

  Rimuove un elemento dalla struttura dati.

- virtual bool InsertAll (const TraversableContainer< Data > &container)

  Inserisce tutti gli elementi da un contenitore attraversabile.
- virtual bool InsertAll (MappableContainer< Data > &&container)

  Inserisce tutti gli elementi da un contenitore mappabile.
- virtual bool RemoveAll (const TraversableContainer< Data > &container)

  Rimuove tutti gli elementi contenuti anche in un altro contenitore.
- virtual bool InsertSome (const TraversableContainer< Data > &container)

  Inserisce alcuni elementi (non necessariamente tutti) da un contenitore.
- virtual bool InsertSome (MappableContainer< Data > &&container)

  Inserisce alcuni elementi (non necessariamente tutti) da un contenitore mappabile.
- virtual bool RemoveSome (const TraversableContainer< Data > &container)

  Rimuove alcuni elementi (non necessariamente tutti) contenuti in un altro contenitore.

Public Member Functions inherited from lasd::TestableContainer< Data >

- virtual ~TestableContainer ()=default

  Distruttore virtuale di default.
- TestableContainer & operator= (const TestableContainer &)=delete
- TestableContainer & operator= (TestableContainer &&) noexcept=delete
- bool operator== (const TestableContainer &) const noexcept=delete

  Operatore di confronto di uguaglianza disabilitato.
- bool operator!= (const TestableContainer &) const noexcept=delete

  Operatore di confronto di disuguaglianza disabilitato.
- virtual bool Exists (const Data &dato) const noexcept=0

  Verifica se un elemento esiste nel contenitore.

Public Member Functions inherited from lasd::Container

- virtual ~Container ()=default

  Destructor.
- Container & operator= (const Container &)=delete

  Copy assignment of abstract types is not possible.
- Container & operator= (Container &&) noexcept=delete

  Move assignment of abstract types is not possible.
- bool operator== (const Container &) const noexcept=delete
- bool operator!= (const Container &) const noexcept=delete
- virtual bool Empty () const noexcept

  La funzione Empty() controlla se la struttura è vuota (concrete function should not throw exceptions)
- virtual ulong Size () const noexcept

  La funzione Size() restituisce il numero di elementi presenti nella struttura (concrete function should not throw exceptions)

Additional Inherited Members

Protected Member Functions inherited from lasd::TestableContainer< Data >

- TestableContainer ()=default

  Costruttore di default protetto.

Protected Member Functions inherited from lasd::Container

- Container ()=default

  Default constructor.

Protected Attributes inherited from lasd::Container

- ulong size = 0

6.3.1 Detailed Description

template<typename Data>
class lasd::DictionaryContainer< Data >

La classe DictionaryContainer definisce al suo interno tutti quei metodi che permettono ad una struttura dati di funzionare come un dizionario.

Definition at line 13 of file dictionary.hpp.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 ~DictionaryContainer()

```
template<typename Data>
virtual lasd::DictionaryContainer< Data >::~DictionaryContainer ()  [virtual], [default]
```

Destructor.

6.3.3 Member Function Documentation

6.3.3.1 Insert() [1/2]

```
template<typename Data>
virtual bool lasd::DictionaryContainer< Data >::Insert (
            const Data & val)  [pure virtual]
```

Inserisce un elemento nella struttura dati.

Parameters

in	*val*	L'elemento da inserire

Returns

true se l'inserimento ha avuto successo, false altrimenti

6.3.3.2 Insert() [2/2]

```
template<typename Data>
virtual bool lasd::DictionaryContainer< Data >::Insert (
            Data && val)  [pure virtual]
```

Inserisce un elemento usando move semantics.

Parameters

in	*val*	L'elemento da inserire (movibile)

Returns

true se l'inserimento ha avuto successo, false altrimenti

6.3.3.3 InsertAll() [1/2]

```
template<typename Data>
bool lasd::DictionaryContainer< Data >::InsertAll (
            const TraversableContainer< Data > & container)  [inline], [virtual]
```

Inserisce tutti gli elementi da un contenitore attraversabile.

Parameters

in	*container*	Il contenitore da cui inserire

Returns

true se almeno un elemento è stato inserito, false altrimenti

Definition at line 4 of file dictionary.cpp.

6.3.3.4 InsertAll() [2/2]

```
template<typename Data>
bool lasd::DictionaryContainer< Data >::InsertAll (
            MappableContainer< Data > && container)  [inline], [virtual]
```

Inserisce tutti gli elementi da un contenitore mappabile.

Parameters

in	*container*	Il contenitore da cui inserire

Returns

true se almeno un elemento è stato inserito, false altrimenti

Definition at line 13 of file dictionary.cpp.

6.3.3.5 InsertSome() [1/2]

```
template<typename Data>
bool lasd::DictionaryContainer< Data >::InsertSome (
            const TraversableContainer< Data > & container)  [inline], [virtual]
```

Inserisce alcuni elementi (non necessariamente tutti) da un contenitore.

Parameters

in	*container*	Il contenitore da cui inserire

Returns

true se almeno un elemento è stato inserito, false altrimenti

Definition at line 31 of file dictionary.cpp.

6.3.3.6 InsertSome() [2/2]

```
template<typename Data>
bool lasd::DictionaryContainer< Data >::InsertSome (
            MappableContainer< Data > && container)  [inline], [virtual]
```

Inserisce alcuni elementi (non necessariamente tutti) da un contenitore mappabile.

Parameters

in	*container*	Il contenitore da cui inserire

Returns

true se almeno un elemento è stato inserito, false altrimenti

Definition at line 40 of file dictionary.cpp.

6.3.3.7 operator"!=()

```
template<typename Data>
bool lasd::DictionaryContainer< Data >::operator!= (
            const DictionaryContainer< Data > & ) const  [delete], [noexcept]
```

6.3.3.8 operator=() [1/2]

```
template<typename Data>
DictionaryContainer & lasd::DictionaryContainer< Data >::operator= (
            const DictionaryContainer< Data > & )  [delete]
```

Copy assignment.

6.3.3.9 operator=() [2/2]

```
template<typename Data>
DictionaryContainer & lasd::DictionaryContainer< Data >::operator= (
            DictionaryContainer< Data > && )  [delete]
```

Move assignment.

6.3.3.10 operator==()

```
template<typename Data>
bool lasd::DictionaryContainer< Data >::operator== (
            const DictionaryContainer< Data > & ) const  [delete], [noexcept]
```

6.3.3.11 Remove()

```
template<typename Data>
virtual bool lasd::DictionaryContainer< Data >::Remove (
            const Data & val)  [pure virtual]
```

Rimuove un elemento dalla struttura dati.

Parameters

in	*val*	L'elemento da rimuovere

Returns

true se la rimozione ha avuto successo, false altrimenti

6.3.3.12 RemoveAll()

```
template<typename Data>
bool lasd::DictionaryContainer< Data >::RemoveAll (
            const TraversableContainer< Data > & container)  [inline], [virtual]
```

Rimuove tutti gli elementi contenuti anche in un altro contenitore.

Parameters

in	*container*	Il contenitore da cui rimuovere

Returns

true se almeno un elemento è stato rimosso, false altrimenti

Definition at line 22 of file dictionary.cpp.

6.3.3.13 RemoveSome()

```
template<typename Data>
bool lasd::DictionaryContainer< Data >::RemoveSome (
            const TraversableContainer< Data > & container)  [inline], [virtual]
```

Rimuove alcuni elementi (non necessariamente tutti) contenuti in un altro contenitore.

Parameters

in	*container*	Il contenitore da cui rimuovere

Returns

true se almeno un elemento è stato rimosso, false altrimenti

Definition at line 49 of file dictionary.cpp.

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

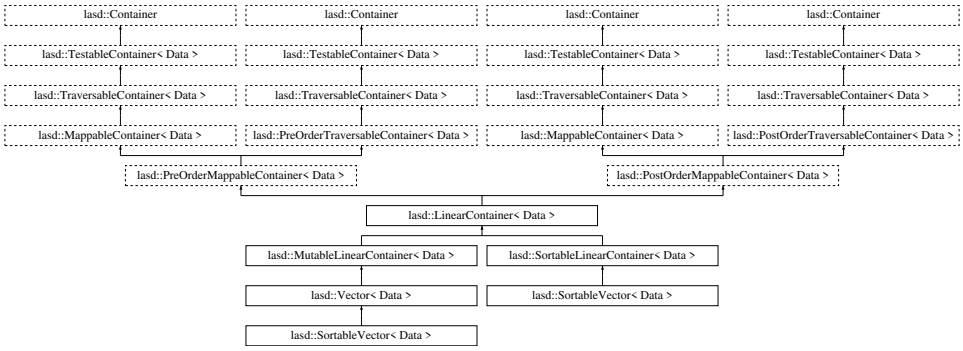
- Exercise1/container/dictionary.hpp
- Exercise1/container/dictionary.cpp

6.4 lasd::LinearContainer< Data > Class Template Reference

Classe astratta che rappresenta un contenitore lineare accessibile per posizione.

`#include <linear.hpp>`

Inheritance diagram for lasd::LinearContainer< Data >:



Public Member Functions

- virtual ~LinearContainer ()=default

  Distruttore virtuale di default.
- LinearContainer & operator= (const LinearContainer &)=delete
- LinearContainer & operator= (LinearContainer &&) noexcept=delete
- bool operator== (const LinearContainer &) const noexcept

  Operatore di confronto di uguaglianza.
- bool operator!= (const LinearContainer &) const noexcept

  Operatore di confronto di disuguaglianza.
- virtual const Data & operator[ ] (const ulong int index) const =0

  Accesso in sola lettura all'elemento in posizione specifica.
- virtual Data & operator[ ] (const ulong int index)=0

  Accesso in lettura/scrittura all'elemento in posizione specifica.
- virtual const Data & Front () const

Restituisce il primo elemento (costante).

- virtual Data & Front ()

  Restituisce il primo elemento.
- virtual const Data & Back () const

  Restituisce l'ultimo elemento (costante).
- virtual Data & Back ()

  Restituisce l'ultimo elemento.
- void Traverse (TraverseFun) const override

  Traversamento in ordine predefinito.
- void PreOrderTraverse (TraverseFun) const override

  Traversamento in ordine PreOrder.
- void PostOrderTraverse (TraverseFun) const override

  Traversamento in ordine PostOrder.
- void Map (MapFun) override

  Mappatura in ordine predefinito.
- void PreOrderMap (MapFun) override

  Mappatura in ordine PreOrder.
- void PostOrderMap (MapFun) override

  Mappatura in ordine PostOrder.

Public Member Functions inherited from lasd::PreOrderMappableContainer< Data >

- virtual ~PreOrderMappableContainer ()=default

  Distruttore virtuale di default.
- PreOrderMappableContainer & operator= (const PreOrderMappableContainer &)=delete
- PreOrderMappableContainer & operator= (PreOrderMappableContainer &&) noexcept=delete
- bool operator== (const PreOrderMappableContainer &) const noexcept=delete

  Operatore di confronto di uguaglianza disabilitato.
- bool operator!= (const PreOrderMappableContainer &) const noexcept=delete

  Operatore di confronto di disuguaglianza disabilitato.
- void Map (MapFun fun) override

  Applica la funzione mappante secondo la strategia PreOrder.

Public Member Functions inherited from lasd::MappableContainer< Data >

- virtual ~MappableContainer () noexcept=default

  Distruttore virtuale di default.
- MappableContainer & operator= (const MappableContainer &) noexcept=delete
- MappableContainer & operator= (MappableContainer &&) noexcept=delete
- bool operator== (const MappableContainer &) const noexcept=delete

  Operatore di confronto di uguaglianza disabilitato.
- bool operator!= (const MappableContainer &) const noexcept=delete

  Operatore di confronto di disuguaglianza disabilitato.

Public Member Functions inherited from lasd::TraversableContainer< Data >

- virtual ~TraversableContainer ()=default

  Distruttore virtuale di default.
- TraversableContainer & operator= (const TraversableContainer &)=delete
- TraversableContainer & operator= (TraversableContainer &&) noexcept=delete
- bool operator== (const TraversableContainer &) const noexcept=delete
- bool operator!= (const TraversableContainer &) const noexcept=delete
- template<typename Accumulator>
  Accumulator Fold (FoldFun< Accumulator > func, Accumulator base) const

  Esegue una riduzione (fold) sul contenitore.
- bool Exists (const Data &elem) const noexcept override

  Verifica se un elemento esiste nel contenitore.

Public Member Functions inherited from lasd::TestableContainer< Data >

- virtual ~TestableContainer ()=default

  Distruttore virtuale di default.
- TestableContainer & operator= (const TestableContainer &)=delete
- TestableContainer & operator= (TestableContainer &&) noexcept=delete
- bool operator== (const TestableContainer &) const noexcept=delete

  Operatore di confronto di uguaglianza disabilitato.
- bool operator!= (const TestableContainer &) const noexcept=delete

  Operatore di confronto di disuguaglianza disabilitato.

Public Member Functions inherited from lasd::Container

- virtual ~Container ()=default

  Destructor.
- Container & operator= (const Container &)=delete

  Copy assignment of abstract types is not possible.
- Container & operator= (Container &&) noexcept=delete

  Move assignment of abstract types is not possible.
- bool operator== (const Container &) const noexcept=delete
- bool operator!= (const Container &) const noexcept=delete
- virtual bool Empty () const noexcept

  La funzione Empty() controlla se la struttura è vuota (concrete function should not throw exceptions)
- virtual ulong Size () const noexcept

  La funzione Size() restituisce il numero di elementi presenti nella struttura (concrete function should not throw exceptions)

Public Member Functions inherited from lasd::PreOrderTraversableContainer< Data >

- virtual ~PreOrderTraversableContainer ()=default

  Distruttore virtuale di default.
- PreOrderTraversableContainer & operator= (const PreOrderTraversableContainer &)=delete
- PreOrderTraversableContainer & operator= (PreOrderTraversableContainer &&) noexcept=delete
- bool operator== (const PreOrderTraversableContainer &) const noexcept=delete
- bool operator!= (const PreOrderTraversableContainer &) const noexcept=delete
- template<typename Accumulator>
  Accumulator PreOrderFold (FoldFun< Accumulator > func, Accumulator base) const

  Esegue una riduzione in pre-ordine.
- void Traverse (TraverseFun func) const override

  Implementazione della traversata base come traversata in pre-ordine.

Public Member Functions inherited from lasd::PostOrderMappableContainer< Data >

- virtual ~PostOrderMappableContainer ()=default

  Distruttore virtuale di default.
- PostOrderMappableContainer & operator= (const PostOrderMappableContainer &)=delete
- PostOrderMappableContainer & operator= (PostOrderMappableContainer &&) noexcept=delete
- bool operator== (const PostOrderMappableContainer &) const noexcept=delete

  Operatore di confronto di uguaglianza disabilitato.
- bool operator!= (const PostOrderMappableContainer &) const noexcept=delete

  Operatore di confronto di disuguaglianza disabilitato.
- void Map (const MapFun fun) override

  Applica la funzione mappante secondo la strategia PostOrder.

Public Member Functions inherited from lasd::PostOrderTraversableContainer< Data >

- virtual ~PostOrderTraversableContainer ()=default

  Distruttore virtuale di default.
- PostOrderTraversableContainer & operator= (const PostOrderTraversableContainer &)=delete
- PostOrderTraversableContainer & operator= (PostOrderTraversableContainer &&) noexcept=delete
- bool operator== (const PostOrderTraversableContainer &) const noexcept=delete
- bool operator!= (const PostOrderTraversableContainer &) const noexcept=delete
- template<typename Accumulator>
  Accumulator PostOrderFold (FoldFun< Accumulator > func, Accumulator base) const

  Esegue una riduzione in post-ordine.
- void Traverse (TraverseFun func) const override

  Implementazione della traversata base come traversata in post-ordine.

Protected Attributes

- ulong size

Protected Attributes inherited from lasd::Container

- ulong size = 0

Additional Inherited Members

Public Types inherited from lasd::MappableContainer< Data >

- using MapFun = std::function<void(Data &)>

  Tipo di funzione mappante: accetta un riferimento modificabile a un elemento.

Public Types inherited from lasd::TraversableContainer< Data >

- using TraverseFun = std::function<void(const Data &)>

  Tipo funzione per la traversata (funzione unaria applicata a ogni dato).
- template<typename Accumulator>
  using FoldFun = std::function<Accumulator(const Data &, const Accumulator &)>

  Tipo funzione per fold (funzione binaria con accumulatore).

Public Types inherited from lasd::PreOrderTraversableContainer< Data >

- template<typename Accumulator>
  using FoldFun = typename TraversableContainer<Data>::FoldFun<Accumulator>

  Tipo funzione per fold in pre-ordine.

Public Types inherited from lasd::PostOrderTraversableContainer< Data >

- template<typename Accumulator>
  using FoldFun = typename TraversableContainer<Data>::FoldFun<Accumulator>

  Tipo funzione per fold in post-ordine.

Protected Member Functions inherited from lasd::TestableContainer< Data >

- TestableContainer ()=default

  Costruttore di default protetto.

Protected Member Functions inherited from lasd::Container

- Container ()=default

  Default constructor.

6.4.1 Detailed Description

template<typename Data>
class lasd::LinearContainer< Data >

Classe astratta che rappresenta un contenitore lineare accessibile per posizione.

Estende sia PreOrder che PostOrder mappabile e traversabile.

Template Parameters

Data	Tipo dei dati contenuti.

Definition at line 17 of file linear.hpp.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 ~LinearContainer()

```
template<typename Data>
virtual lasd::LinearContainer< Data >::~LinearContainer ()  [virtual], [default]
```

Distruttore virtuale di default.

6.4.3 Member Function Documentation

6.4.3.1 Back() [1/2]

```
template<typename Data>
Data & lasd::LinearContainer< Data >::Back ()  [inline], [virtual]
```

Restituisce l'ultimo elemento.

Returns

Riferimento all'ultimo elemento.

Reimplemented in lasd::MutableLinearContainer< Data >, and lasd::Vector< Data >.

Definition at line 63 of file linear.cpp.

6.4.3.2 Back() [2/2]

```
template<typename Data>
const Data & lasd::LinearContainer< Data >::Back () const  [inline], [virtual]
```

Restituisce l'ultimo elemento (costante).

Returns

Riferimento costante all'ultimo elemento.

Reimplemented in lasd::Vector< Data >.

Definition at line 53 of file linear.cpp.

6.4.3.3 Front() [1/2]

```
template<typename Data>
Data & lasd::LinearContainer< Data >::Front ()  [inline], [virtual]
```

Restituisce il primo elemento.

Returns

Riferimento al primo elemento.

Reimplemented in lasd::MutableLinearContainer< Data >, and lasd::Vector< Data >.

Definition at line 43 of file linear.cpp.

6.4.3.4 Front() [2/2]

```
template<typename Data>
const Data & lasd::LinearContainer< Data >::Front () const  [inline], [virtual]
```

Restituisce il primo elemento (costante).

Returns

Riferimento costante al primo elemento.

Reimplemented in lasd::Vector< Data >.

Definition at line 33 of file linear.cpp.

6.4.3.5 Map()

```
template<typename Data>
void lasd::LinearContainer< Data >::Map (
            MapFun func)  [inline], [override], [virtual]
```

Mappatura in ordine predefinito.

Implements lasd::MappableContainer< Data >.

Reimplemented in lasd::MutableLinearContainer< Data >.

Definition at line 101 of file linear.cpp.

6.4.3.6 operator"!=()

```
template<typename Data>
bool lasd::LinearContainer< Data >::operator!= (
            const LinearContainer< Data > & con) const  [inline], [noexcept]
```

Operatore di confronto di disuguaglianza.

Definition at line 25 of file linear.cpp.

6.4.3.7 operator=() [1/2]

```
template<typename Data>
LinearContainer & lasd::LinearContainer< Data >::operator= (
            const LinearContainer< Data > & )  [delete]
```

6.4.3.8 operator=() [2/2]

```
template<typename Data>
LinearContainer & lasd::LinearContainer< Data >::operator= (
            LinearContainer< Data > && )  [delete], [noexcept]
```

6.4.3.9 operator==()

```
template<typename Data>
bool lasd::LinearContainer< Data >::operator== (
            const LinearContainer< Data > & con) const  [inline], [noexcept]
```

Operatore di confronto di uguaglianza.

Definition at line 6 of file linear.cpp.

6.4.3.10 operator[]() [1/2]

```
template<typename Data>
virtual const Data & lasd::LinearContainer< Data >::operator[] (
            const ulong int index) const  [pure virtual]
```

Accesso in sola lettura all'elemento in posizione specifica.

Parameters

index	Posizione dell'elemento.

Returns

Riferimento costante all'elemento.

6.4.3.11 operator[]() [2/2]

```
template<typename Data>
virtual Data & lasd::LinearContainer< Data >::operator[] (
            const ulong int index)  [pure virtual]
```

Accesso in lettura/scrittura all'elemento in posizione specifica.

Parameters

index	Posizione dell'elemento.

Returns

Riferimento all'elemento.

6.4.3.12 PostOrderMap()

```
template<typename Data>
void lasd::LinearContainer< Data >::PostOrderMap (
            MapFun func)  [inline], [override], [virtual]
```

Mappatura in ordine PostOrder.

Implements lasd::PostOrderMappableContainer< Data >.

Reimplemented in lasd::MutableLinearContainer< Data >.

Definition at line 116 of file linear.cpp.

6.4.3.13 PostOrderTraverse()

```
template<typename Data>
void lasd::LinearContainer< Data >::PostOrderTraverse (
            TraverseFun func) const  [inline], [override], [virtual]
```

Traversamento in ordine PostOrder.

Implements lasd::PostOrderTraversableContainer< Data >.

Definition at line 91 of file linear.cpp.

6.4.3.14 PreOrderMap()

```
template<typename Data>
void lasd::LinearContainer< Data >::PreOrderMap (
            MapFun func)  [inline], [override], [virtual]
```

Mappatura in ordine PreOrder.

Implements lasd::PreOrderMappableContainer< Data >.

Reimplemented in lasd::MutableLinearContainer< Data >.

Definition at line 107 of file linear.cpp.

6.4.3.15 PreOrderTraverse()

```
template<typename Data>
void lasd::LinearContainer< Data >::PreOrderTraverse (
            TraverseFun func) const  [inline], [override], [virtual]
```

Traversamento in ordine PreOrder.

Implements lasd::PreOrderTraversableContainer< Data >.

Definition at line 81 of file linear.cpp.

6.4.3.16 Traverse()

```
template<typename Data>
void lasd::LinearContainer< Data >::Traverse (
            TraverseFun func) const  [inline], [override], [virtual]
```

Traversamento in ordine predefinito.

Implements lasd::TraversableContainer< Data >.

Definition at line 74 of file linear.cpp.

6.4.4 Member Data Documentation

6.4.4.1 size

```
template<typename Data>
ulong lasd::Container::size  [protected]
```

L'attributo size indica il numero di elementi presenti nel Container

Definition at line 15 of file container.hpp.

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

- Exercise1/container/linear.hpp
- Exercise1/container/linear.cpp

6.5 lasd::List< Data > Class Template Reference

```
#include <list.hpp>
```

Classes

- struct Node

6.5.1 Detailed Description

template<typename Data>
class lasd::List< Data >

Definition at line 16 of file list.hpp.

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

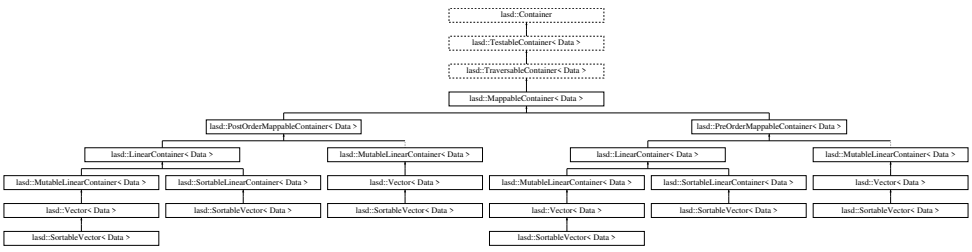
- Exercise1/list/list.hpp

6.6 lasd::MappableContainer< Data > Class Template Reference

Classe astratta che estende TraversableContainer, permettendo la modifica degli elementi tramite funzioni mappanti.

```
#include <mappable.hpp>
```

Inheritance diagram for lasd::MappableContainer< Data >:



Public Types

- using MapFun = std::function<void(Data &)>

  Tipo di funzione mappante: accetta un riferimento modificabile a un elemento.

Public Types inherited from lasd::TraversableContainer< Data >

- using TraverseFun = std::function<void(const Data &)>

  Tipo funzione per la traversata (funzione unaria applicata a ogni dato).
- template<typename Accumulator>
  using FoldFun = std::function<Accumulator(const Data &, const Accumulator &)>

  Tipo funzione per fold (funzione binaria con accumulatore).

Public Member Functions

- virtual ~MappableContainer () noexcept=default

  Distruttore virtuale di default.
- MappableContainer & operator= (const MappableContainer &) noexcept=delete
- MappableContainer & operator= (MappableContainer &&) noexcept=delete
- bool operator== (const MappableContainer &) const noexcept=delete

  Operatore di confronto di uguaglianza disabilitato.
- bool operator!= (const MappableContainer &) const noexcept=delete

  Operatore di confronto di disuguaglianza disabilitato.
- virtual void Map (const MapFun fun)=0

  Applica la funzione specificata a ciascun elemento del contenitore.

Public Member Functions inherited from lasd::TraversableContainer< Data >

- virtual ~TraversableContainer ()=default

  Distruttore virtuale di default.
- TraversableContainer & operator= (const TraversableContainer &)=delete
- TraversableContainer & operator= (TraversableContainer &&) noexcept=delete
- bool operator== (const TraversableContainer &) const noexcept=delete
- bool operator!= (const TraversableContainer &) const noexcept=delete
- virtual void Traverse (TraverseFun func) const =0

  Esegue una funzione su ogni elemento del contenitore.
- template<typename Accumulator>
  Accumulator Fold (FoldFun< Accumulator > func, Accumulator base) const

  Esegue una riduzione (fold) sul contenitore.
- bool Exists (const Data &elem) const noexcept override

  Verifica se un elemento esiste nel contenitore.

Public Member Functions inherited from lasd::TestableContainer< Data >

- virtual ~TestableContainer ()=default

  Distruttore virtuale di default.
- TestableContainer & operator= (const TestableContainer &)=delete
- TestableContainer & operator= (TestableContainer &&) noexcept=delete
- bool operator== (const TestableContainer &) const noexcept=delete

  Operatore di confronto di uguaglianza disabilitato.
- bool operator!= (const TestableContainer &) const noexcept=delete

  Operatore di confronto di disuguaglianza disabilitato.

Public Member Functions inherited from lasd::Container

- virtual ~Container ()=default

  Destructor.
- Container & operator= (const Container &)=delete

  Copy assignment of abstract types is not possible.
- Container & operator= (Container &&) noexcept=delete

  Move assignment of abstract types is not possible.
- bool operator== (const Container &) const noexcept=delete
- bool operator!= (const Container &) const noexcept=delete
- virtual bool Empty () const noexcept

  La funzione Empty() controlla se la struttura è vuota (concrete function should not throw exceptions)
- virtual ulong Size () const noexcept

  La funzione Size() restituisce il numero di elementi presenti nella struttura (concrete function should not throw exceptions)

Additional Inherited Members

Protected Member Functions inherited from lasd::TestableContainer< Data >

- TestableContainer ()=default

  Costruttore di default protetto.

Protected Member Functions inherited from lasd::Container

- Container ()=default

  Default constructor.

Protected Attributes inherited from lasd::Container

- ulong size = 0

6.6.1 Detailed Description

template<typename Data>
class lasd::MappableContainer< Data >

Classe astratta che estende TraversableContainer, permettendo la modifica degli elementi tramite funzioni mappanti.

Template Parameters

Data	Tipo dei dati contenuti.

Definition at line 16 of file mappable.hpp.

6.6.2 Member Typedef Documentation

6.6.2.1 MapFun

```
template<typename Data>
using lasd::MappableContainer< Data >::MapFun = std::function<void(Data &)>
```

Tipo di funzione mappante: accetta un riferimento modificabile a un elemento.

Definition at line 38 of file mappable.hpp.

6.6.3 Constructor & Destructor Documentation

6.6.3.1 ~MappableContainer()

```
template<typename Data>
virtual lasd::MappableContainer< Data >::~MappableContainer ()  [virtual], [default], [noexcept]
```

Distruttore virtuale di default.

6.6.4 Member Function Documentation

6.6.4.1 Map()

```
template<typename Data>
virtual void lasd::MappableContainer< Data >::Map (
            const MapFun fun)  [pure virtual]
```

Applica la funzione specificata a ciascun elemento del contenitore.

Parameters

fun	Funzione che modifica ogni elemento tramite riferimento.

Implemented in lasd::LinearContainer< Data >, lasd::MutableLinearContainer< Data >, lasd::PostOrderMappableContainer< Data > and lasd::PreOrderMappableContainer< Data >.

6.6.4.2 operator"!=()

```
template<typename Data>
bool lasd::MappableContainer< Data >::operator!= (
            const MappableContainer< Data > & ) const  [delete], [noexcept]
```

Operatore di confronto di disuguaglianza disabilitato.

6.6.4.3 operator=() [1/2]

```
template<typename Data>
MappableContainer & lasd::MappableContainer< Data >::operator= (
            const MappableContainer< Data > & )  [delete], [noexcept]
```

6.6.4.4 operator=() [2/2]

```
template<typename Data>
MappableContainer & lasd::MappableContainer< Data >::operator= (
            MappableContainer< Data > && )  [delete], [noexcept]
```

6.6.4.5 operator==()

```
template<typename Data>
bool lasd::MappableContainer< Data >::operator== (
            const MappableContainer< Data > & ) const  [delete], [noexcept]
```

Operatore di confronto di uguaglianza disabilitato.

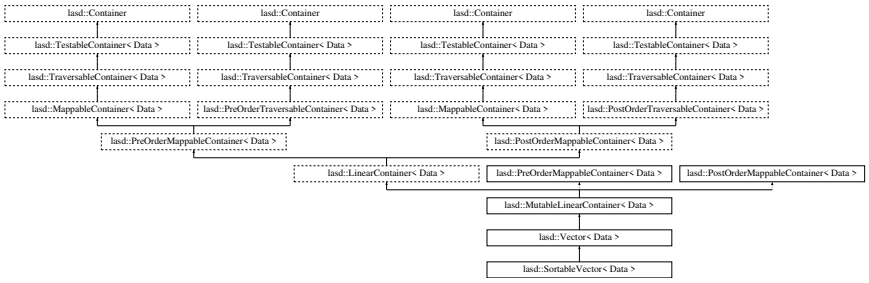
The documentation for this class was generated from the following file:

- Exercise1/container/mappable.hpp

6.7 lasd::MutableLinearContainer< Data > Class Template Reference

```
#include <linear.hpp>
```

Inheritance diagram for lasd::MutableLinearContainer< Data >:



Public Member Functions

- virtual ~MutableLinearContainer ()=default

  Distruttore virtuale di default.
- MutableLinearContainer & operator= (const MutableLinearContainer &)=delete
- MutableLinearContainer & operator= (MutableLinearContainer &&) noexcept=delete
- virtual Data & operator[ ] (const ulong index)=0

  Restituisce un riferimento modificabile all'elemento in posizione specifica.
- virtual Data & Front ()=0

  Restituisce un riferimento modificabile al primo elemento del contenitore.
- virtual Data & Back ()=0

  Restituisce un riferimento modificabile all'ultimo elemento del contenitore.
- void Map (MapFun mapFun) override

  Applica una funzione a ciascun elemento (mappatura) in ordine indefinito.
- virtual void PreOrderMap (MapFun mapFun) override=0

  Applica una funzione a ciascun elemento (mappatura) in ordine PreOrder.
- virtual void PostOrderMap (MapFun mapFun) override=0

  Applica una funzione a ciascun elemento (mappatura) in ordine PostOrder.

Public Member Functions inherited from lasd::LinearContainer< Data >

- virtual ~LinearContainer ()=default

  Distruttore virtuale di default.
- LinearContainer & operator= (const LinearContainer &)=delete
- LinearContainer & operator= (LinearContainer &&) noexcept=delete
- bool operator== (const LinearContainer &) const noexcept

  Operatore di confronto di uguaglianza.
- bool operator!= (const LinearContainer &) const noexcept

  Operatore di confronto di disuguaglianza.
- virtual const Data & operator[ ] (const ulong int index) const =0

  Accesso in sola lettura all'elemento in posizione specifica.
- virtual Data & operator[ ] (const ulong int index)=0

  Accesso in lettura/scrittura all'elemento in posizione specifica.
- virtual const Data & Front () const

  Restituisce il primo elemento (costante).
- virtual const Data & Back () const

  Restituisce l'ultimo elemento (costante).
- void Traverse (TraverseFun) const override

  Traversamento in ordine predefinito.
- void PreOrderTraverse (TraverseFun) const override

  Traversamento in ordine PreOrder.
- void PostOrderTraverse (TraverseFun) const override

  Traversamento in ordine PostOrder.

Public Member Functions inherited from lasd::PreOrderMappableContainer< Data >

- virtual ~PreOrderMappableContainer ()=default

  Distruttore virtuale di default.
- PreOrderMappableContainer & operator= (const PreOrderMappableContainer &)=delete
- PreOrderMappableContainer & operator= (PreOrderMappableContainer &&) noexcept=delete
- bool operator== (const PreOrderMappableContainer &) const noexcept=delete

  Operatore di confronto di uguaglianza disabilitato.
- bool operator!= (const PreOrderMappableContainer &) const noexcept=delete

  Operatore di confronto di disuguaglianza disabilitato.
- void Map (MapFun fun) override

  Applica la funzione mappante secondo la strategia PreOrder.

Public Member Functions inherited from lasd::MappableContainer< Data >

- virtual ~MappableContainer () noexcept=default

  Distruttore virtuale di default.
- MappableContainer & operator= (const MappableContainer &) noexcept=delete
- MappableContainer & operator= (MappableContainer &&) noexcept=delete
- bool operator== (const MappableContainer &) const noexcept=delete

  Operatore di confronto di uguaglianza disabilitato.
- bool operator!= (const MappableContainer &) const noexcept=delete

  Operatore di confronto di disuguaglianza disabilitato.

Public Member Functions inherited from lasd::TraversableContainer< Data >

- virtual ~TraversableContainer ()=default

  Distruttore virtuale di default.
- TraversableContainer & operator= (const TraversableContainer &)=delete
- TraversableContainer & operator= (TraversableContainer &&) noexcept=delete
- bool operator== (const TraversableContainer &) const noexcept=delete
- bool operator!= (const TraversableContainer &) const noexcept=delete
- template<typename Accumulator>
  Accumulator Fold (FoldFun< Accumulator > func, Accumulator base) const

  Esegue una riduzione (fold) sul contenitore.
- bool Exists (const Data &elem) const noexcept override

  Verifica se un elemento esiste nel contenitore.

Public Member Functions inherited from lasd::TestableContainer< Data >

- virtual ~TestableContainer ()=default

  Distruttore virtuale di default.
- TestableContainer & operator= (const TestableContainer &)=delete
- TestableContainer & operator= (TestableContainer &&) noexcept=delete
- bool operator== (const TestableContainer &) const noexcept=delete

  Operatore di confronto di uguaglianza disabilitato.
- bool operator!= (const TestableContainer &) const noexcept=delete

  Operatore di confronto di disuguaglianza disabilitato.

Public Member Functions inherited from lasd::Container

- virtual ~Container ()=default

  Destructor.
- Container & operator= (const Container &)=delete

  Copy assignment of abstract types is not possible.
- Container & operator= (Container &&) noexcept=delete

  Move assignment of abstract types is not possible.
- bool operator== (const Container &) const noexcept=delete
- bool operator!= (const Container &) const noexcept=delete
- virtual bool Empty () const noexcept

  La funzione Empty() controlla se la struttura è vuota (concrete function should not throw exceptions)
- virtual ulong Size () const noexcept

  La funzione Size() restituisce il numero di elementi presenti nella struttura (concrete function should not throw exceptions)

Public Member Functions inherited from lasd::PreOrderTraversableContainer< Data >

- virtual ~PreOrderTraversableContainer ()=default

  Distruttore virtuale di default.
- PreOrderTraversableContainer & operator= (const PreOrderTraversableContainer &)=delete
- PreOrderTraversableContainer & operator= (PreOrderTraversableContainer &&) noexcept=delete
- bool operator== (const PreOrderTraversableContainer &) const noexcept=delete
- bool operator!= (const PreOrderTraversableContainer &) const noexcept=delete
- template<typename Accumulator>
  Accumulator PreOrderFold (FoldFun< Accumulator > func, Accumulator base) const

  Esegue una riduzione in pre-ordine.
- void Traverse (TraverseFun func) const override

  Implementazione della traversata base come traversata in pre-ordine.

Public Member Functions inherited from lasd::PostOrderMappableContainer< Data >

- virtual ~PostOrderMappableContainer ()=default

  Distruttore virtuale di default.
- PostOrderMappableContainer & operator= (const PostOrderMappableContainer &)=delete
- PostOrderMappableContainer & operator= (PostOrderMappableContainer &&) noexcept=delete
- bool operator== (const PostOrderMappableContainer &) const noexcept=delete

  Operatore di confronto di uguaglianza disabilitato.
- bool operator!= (const PostOrderMappableContainer &) const noexcept=delete

  Operatore di confronto di disuguaglianza disabilitato.
- void Map (const MapFun fun) override

  Applica la funzione mappante secondo la strategia PostOrder.

Public Member Functions inherited from lasd::PostOrderTraversableContainer< Data >

- virtual ~PostOrderTraversableContainer ()=default

  Distruttore virtuale di default.
- PostOrderTraversableContainer & operator= (const PostOrderTraversableContainer &)=delete
- PostOrderTraversableContainer & operator= (PostOrderTraversableContainer &&) noexcept=delete
- bool operator== (const PostOrderTraversableContainer &) const noexcept=delete
- bool operator!= (const PostOrderTraversableContainer &) const noexcept=delete
- template<typename Accumulator>
  Accumulator PostOrderFold (FoldFun< Accumulator > func, Accumulator base) const

  Esegue una riduzione in post-ordine.
- void Traverse (TraverseFun func) const override

  Implementazione della traversata base come traversata in post-ordine.

Additional Inherited Members

Public Types inherited from lasd::MappableContainer< Data >

- using MapFun = std::function<void(Data &)>

  Tipo di funzione mappante: accetta un riferimento modificabile a un elemento.

Public Types inherited from lasd::TraversableContainer< Data >

- using TraverseFun = std::function<void(const Data &)>

  Tipo funzione per la traversata (funzione unaria applicata a ogni dato).
- template<typename Accumulator>
  using FoldFun = std::function<Accumulator(const Data &, const Accumulator &)>

  Tipo funzione per fold (funzione binaria con accumulatore).

Public Types inherited from lasd::PreOrderTraversableContainer< Data >

- template<typename Accumulator>
  using FoldFun = typename TraversableContainer<Data>::FoldFun<Accumulator>

  Tipo funzione per fold in pre-ordine.

Public Types inherited from lasd::PostOrderTraversableContainer< Data >

- template<typename Accumulator>
  using FoldFun = typename TraversableContainer<Data>::FoldFun<Accumulator>

  Tipo funzione per fold in post-ordine.

Protected Member Functions inherited from lasd::TestableContainer< Data >

- TestableContainer ()=default

  Costruttore di default protetto.

Protected Member Functions inherited from lasd::Container

- Container ()=default

  Default constructor.

Protected Attributes inherited from lasd::LinearContainer< Data >

- ulong size

Protected Attributes inherited from lasd::Container

- ulong size = 0

6.7.1 Detailed Description

template<typename Data>
class lasd::MutableLinearContainer< Data >

Definition at line 104 of file linear.hpp.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 ~MutableLinearContainer()

```
template<typename Data>
virtual lasd::MutableLinearContainer< Data >::~MutableLinearContainer ()  [virtual], [default]
```

Distruttore virtuale di default.

6.7.3 Member Function Documentation

6.7.3.1 Back()

```
template<typename Data>
virtual Data & lasd::MutableLinearContainer< Data >::Back ()  [pure virtual]
```

Restituisce un riferimento modificabile all'ultimo elemento del contenitore.

Returns

Riferimento all'ultimo elemento.

Exceptions

std::length_error	se il contenitore è vuoto.

Reimplemented from lasd::LinearContainer< Data >.

Implemented in lasd::Vector< Data >.

6.7.3.2 Front()

```
template<typename Data>
virtual Data & lasd::MutableLinearContainer< Data >::Front ()  [pure virtual]
```

Restituisce un riferimento modificabile al primo elemento del contenitore.

Returns

Riferimento al primo elemento.

Exceptions

std::length_error	se il contenitore è vuoto.

Reimplemented from lasd::LinearContainer< Data >.

Implemented in lasd::Vector< Data >.

6.7.3.3 Map()

```
template<typename Data>
void lasd::MutableLinearContainer< Data >::Map (
            MapFun mapFun)  [inline], [override], [virtual]
```

Applica una funzione a ciascun elemento (mappatura) in ordine indefinito.

Parameters

mapFun	Funzione da applicare a ogni elemento.

Reimplemented from lasd::LinearContainer< Data >.

Definition at line 149 of file linear.hpp.

6.7.3.4 operator=() [1/2]

```
template<typename Data>
MutableLinearContainer & lasd::MutableLinearContainer< Data >::operator= (
            const MutableLinearContainer< Data > & )  [delete]
```

6.7.3.5 operator=() [2/2]

```
template<typename Data>
MutableLinearContainer & lasd::MutableLinearContainer< Data >::operator= (
            MutableLinearContainer< Data > && )  [delete], [noexcept]
```

6.7.3.6 operator[]()

```
template<typename Data>
virtual Data & lasd::MutableLinearContainer< Data >::operator[] (
            const ulong index)  [pure virtual]
```

Restituisce un riferimento modificabile all'elemento in posizione specifica.

Parameters

index	Indice dell'elemento da accedere.

Returns

Riferimento all'elemento.

Exceptions

std::out_of_range	se l'indice è fuori dal range.

Implemented in lasd::Vector< Data >.

6.7.3.7 PostOrderMap()

```
template<typename Data>
virtual void lasd::MutableLinearContainer< Data >::PostOrderMap (
            MapFun mapFun)  [override], [pure virtual]
```

Applica una funzione a ciascun elemento (mappatura) in ordine PostOrder.

Parameters

mapFun	Funzione da applicare a ogni elemento.

Reimplemented from lasd::LinearContainer< Data >.

6.7.3.8 PreOrderMap()

```
template<typename Data>
virtual void lasd::MutableLinearContainer< Data >::PreOrderMap (
            MapFun mapFun)  [override], [pure virtual]
```

Applica una funzione a ciascun elemento (mappatura) in ordine PreOrder.

Parameters

mapFun	Funzione da applicare a ogni elemento.

Reimplemented from lasd::LinearContainer< Data >.

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

- Exercise1/container/linear.hpp

6.8 lasd::List< Data >::Node Struct Reference

```
#include <list.hpp>
```

6.8.1 Detailed Description

template<typename Data>
struct lasd::List< Data >::Node

Definition at line 28 of file list.hpp.

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

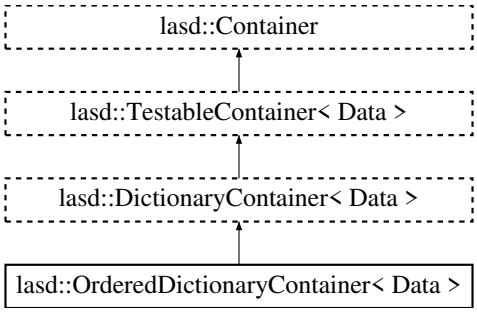
- Exercise1/list/list.hpp

6.9 lasd::OrderedDictionaryContainer< Data > Class Template Reference

Classe che estende DictionaryContainer con capacità di ordinamento.

`#include <dictionary.hpp>`

Inheritance diagram for lasd::OrderedDictionaryContainer< Data >:



Public Member Functions

- virtual ~OrderedDictionaryContainer ()=default

  Destructor.
- OrderedDictionaryContainer & operator= (const OrderedDictionaryContainer &)=delete

  Copy assignment.
- OrderedDictionaryContainer & operator= (OrderedDictionaryContainer &&)=delete

  Move assignment.
- bool operator== (const OrderedDictionaryContainer &) const noexcept=delete
- bool operator!= (const OrderedDictionaryContainer &) const noexcept=delete
- virtual const Data & Min () const =0

  Restituisce il minimo elemento.
- virtual Data MinNRemove ()=0

  Rimuove e restituisce il minimo elemento.
- virtual void RemoveMin ()=0

  Rimuove il minimo elemento.
- virtual const Data & Max () const =0

  Restituisce il massimo elemento.
- virtual Data MaxNRemove ()=0

Rimuove e restituisce il massimo elemento.

- virtual void RemoveMax ()=0

  Rimuove il massimo elemento.
- virtual const Data & Predecessor (const Data &val) const =0

  Restituisce il predecessore di un elemento.
- virtual Data PredecessorNRemove (const Data &val)=0

  Rimuove e restituisce il predecessore di un elemento.
- virtual void RemovePredecessor (const Data &val)=0

  Rimuove il predecessore di un elemento.
- virtual const Data & Successor (const Data &val) const =0

  Restituisce il successore di un elemento.
- virtual Data SuccessorNRemove (const Data &val)=0

  Rimuove e restituisce il successore di un elemento.
- virtual void RemoveSuccessor (const Data &val)=0

  Rimuove il successore di un elemento.

Public Member Functions inherited from lasd::DictionaryContainer< Data >

- virtual ~DictionaryContainer ()=default

  Destructor.
- DictionaryContainer & operator= (const DictionaryContainer &)=delete

  Copy assignment.
- DictionaryContainer & operator= (DictionaryContainer &&)=delete

  Move assignment.
- bool operator== (const DictionaryContainer &) const noexcept=delete
- bool operator!= (const DictionaryContainer &) const noexcept=delete
- virtual bool Insert (const Data &val)=0

  Inserisce un elemento nella struttura dati.
- virtual bool Insert (Data &&val)=0

  Inserisce un elemento usando move semantics.
- virtual bool Remove (const Data &val)=0

  Rimuove un elemento dalla struttura dati.
- virtual bool InsertAll (const TraversableContainer< Data > &container)

  Inserisce tutti gli elementi da un contenitore attraversabile.
- virtual bool InsertAll (MappableContainer< Data > &&container)

  Inserisce tutti gli elementi da un contenitore mappabile.
- virtual bool RemoveAll (const TraversableContainer< Data > &container)

  Rimuove tutti gli elementi contenuti anche in un altro contenitore.
- virtual bool InsertSome (const TraversableContainer< Data > &container)

  Inserisce alcuni elementi (non necessariamente tutti) da un contenitore.
- virtual bool InsertSome (MappableContainer< Data > &&container)

  Inserisce alcuni elementi (non necessariamente tutti) da un contenitore mappabile.
- virtual bool RemoveSome (const TraversableContainer< Data > &container)

  Rimuove alcuni elementi (non necessariamente tutti) contenuti in un altro contenitore.

Public Member Functions inherited from lasd::TestableContainer< Data >

- virtual ~TestableContainer ()=default

  Distruttore virtuale di default.
- TestableContainer & operator= (const TestableContainer &)=delete
- TestableContainer & operator= (TestableContainer &&) noexcept=delete
- bool operator== (const TestableContainer &) const noexcept=delete

  Operatore di confronto di uguaglianza disabilitato.
- bool operator!= (const TestableContainer &) const noexcept=delete

  Operatore di confronto di disuguaglianza disabilitato.
- virtual bool Exists (const Data &dato) const noexcept=0

  Verifica se un elemento esiste nel contenitore.

Public Member Functions inherited from lasd::Container

- virtual ~Container ()=default

  Destructor.
- Container & operator= (const Container &)=delete

  Copy assignment of abstract types is not possible.
- Container & operator= (Container &&) noexcept=delete

  Move assignment of abstract types is not possible.
- bool operator== (const Container &) const noexcept=delete
- bool operator!= (const Container &) const noexcept=delete
- virtual bool Empty () const noexcept

  La funzione Empty() controlla se la struttura è vuota (concrete function should not throw exceptions)
- virtual ulong Size () const noexcept

  La funzione Size() restituisce il numero di elementi presenti nella struttura (concrete function should not throw exceptions)

Additional Inherited Members

Protected Member Functions inherited from lasd::TestableContainer< Data >

- TestableContainer ()=default

  Costruttore di default protetto.

Protected Member Functions inherited from lasd::Container

- Container ()=default

  Default constructor.

Protected Attributes inherited from lasd::Container

- ulong size = 0

6.9.1 Detailed Description

template<typename Data>
class lasd::OrderedDictionaryContainer< Data >

Classe che estende DictionaryContainer con capacità di ordinamento.

Template Parameters

Data	Tipo dei dati contenuti

Definition at line 101 of file dictionary.hpp.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 ~OrderedDictionaryContainer()

```
template<typename Data>
virtual lasd::OrderedDictionaryContainer< Data >::~OrderedDictionaryContainer ()  [virtual],
[default]
```

Destructor.

6.9.3 Member Function Documentation

6.9.3.1 Max()

```
template<typename Data>
virtual const Data & lasd::OrderedDictionaryContainer< Data >::Max () const  [pure virtual]
```

Restituisce il massimo elemento.

Returns

Riferimento costante al massimo elemento

Exceptions

std::length_error	se la struttura è vuota

6.9.3.2 MaxNRemove()

```
template<typename Data>
virtual Data lasd::OrderedDictionaryContainer< Data >::MaxNRemove ()  [pure virtual]
```

Rimuove e restituisce il massimo elemento.

Returns

Il massimo elemento rimosso

Exceptions

std::length_error	se la struttura è vuota

6.9.3.3 Min()

```
template<typename Data>
virtual const Data & lasd::OrderedDictionaryContainer< Data >::Min () const  [pure virtual]
```

Restituisce il minimo elemento.

Returns

Riferimento costante al minimo elemento

Exceptions

std::length_error	se la struttura è vuota

6.9.3.4 MinNRemove()

```
template<typename Data>
virtual Data lasd::OrderedDictionaryContainer< Data >::MinNRemove ()  [pure virtual]
```

Rimuove e restituisce il minimo elemento.

Returns

Il minimo elemento rimosso

Exceptions

std::length_error	se la struttura è vuota

6.9.3.5 operator"!=()

```
template<typename Data>
bool lasd::OrderedDictionaryContainer< Data >::operator!= (
            const OrderedDictionaryContainer< Data > & ) const  [delete], [noexcept]
```

6.9.3.6 operator=() [1/2]

```
template<typename Data>
OrderedDictionaryContainer & lasd::OrderedDictionaryContainer< Data >::operator= (
            const OrderedDictionaryContainer< Data > & )  [delete]
```

Copy assignment.

6.9.3.7 operator=() [2/2]

```
template<typename Data>
OrderedDictionaryContainer & lasd::OrderedDictionaryContainer< Data >::operator= (
            OrderedDictionaryContainer< Data > && )  [delete]
```

Move assignment.

6.9.3.8 operator==()

```
template<typename Data>
bool lasd::OrderedDictionaryContainer< Data >::operator== (
            const OrderedDictionaryContainer< Data > & ) const  [delete], [noexcept]
```

6.9.3.9 Predecessor()

```
template<typename Data>
virtual const Data & lasd::OrderedDictionaryContainer< Data >::Predecessor (
            const Data & val) const  [pure virtual]
```

Restituisce il predecessore di un elemento.

Parameters

in	*val*	L'elemento di riferimento

Returns

Riferimento costante al predecessore

Exceptions

std::length_error	se non esiste un predecessore

6.9.3.10 PredecessorNRemove()

```
template<typename Data>
virtual Data lasd::OrderedDictionaryContainer< Data >::PredecessorNRemove (
            const Data & val)  [pure virtual]
```

Rimuove e restituisce il predecessore di un elemento.

Parameters

in	*val*	L'elemento di riferimento

Returns

Il predecessore rimosso

Exceptions

std::length_error	se non esiste un predecessore

6.9.3.11 RemoveMax()

```
template<typename Data>
virtual void lasd::OrderedDictionaryContainer< Data >::RemoveMax ()  [pure virtual]
```

Rimuove il massimo elemento.

Exceptions

std::length_error	se la struttura è vuota

6.9.3.12 RemoveMin()

```
template<typename Data>
virtual void lasd::OrderedDictionaryContainer< Data >::RemoveMin ()  [pure virtual]
```

Rimuove il minimo elemento.

Exceptions

std::length_error	se la struttura è vuota

6.9.3.13 RemovePredecessor()

```
template<typename Data>
virtual void lasd::OrderedDictionaryContainer< Data >::RemovePredecessor (
            const Data & val)  [pure virtual]
```

Rimuove il predecessore di un elemento.

Parameters

in	*val*	L'elemento di riferimento

Exceptions

std::length_error	se non esiste un predecessore

6.9.3.14 RemoveSuccessor()

```
template<typename Data>
virtual void lasd::OrderedDictionaryContainer< Data >::RemoveSuccessor (
            const Data & val)  [pure virtual]
```

Rimuove il successore di un elemento.

Parameters

in	*val*	L'elemento di riferimento

Exceptions

std::length_error	se non esiste un successore

6.9.3.15 Successor()

```
template<typename Data>
virtual const Data & lasd::OrderedDictionaryContainer< Data >::Successor (
            const Data & val) const  [pure virtual]
```

Restituisce il successore di un elemento.

Parameters

in	*val*	L'elemento di riferimento

Returns

Riferimento costante al successore

Exceptions

std::length_error	se non esiste un successore

6.9.3.16 SuccessorNRemove()

```
template<typename Data>
virtual Data lasd::OrderedDictionaryContainer< Data >::SuccessorNRemove (
            const Data & val)  [pure virtual]
```

Rimuove e restituisce il successore di un elemento.

Parameters

in	*val*	L'elemento di riferimento

Returns

Il successore rimosso

Exceptions

std::length_error	se non esiste un successore

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

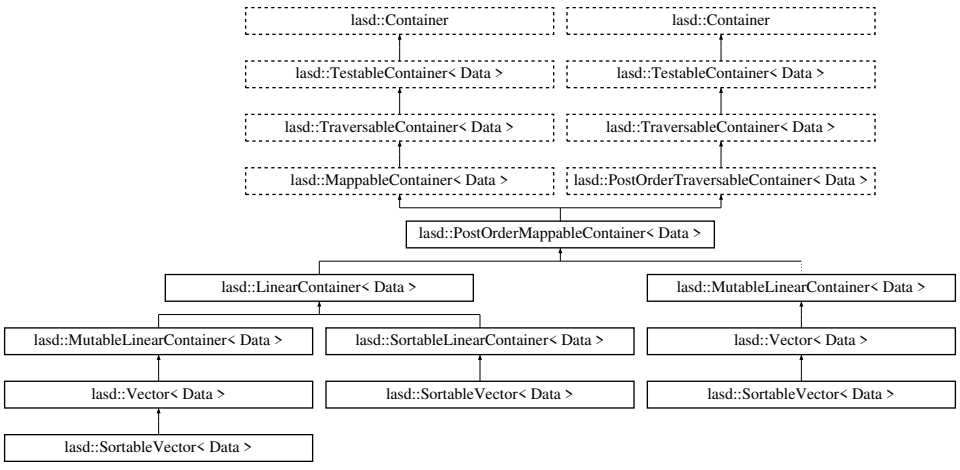
- Exercise1/container/dictionary.hpp

6.10 lasd::PostOrderMappableContainer< Data > Class Template Reference

Estensione di MappableContainer che specifica l'ordine PostOrder per la mappatura.

```
#include <mappable.hpp>
```

Inheritance diagram for lasd::PostOrderMappableContainer< Data >:



Public Member Functions

- virtual ~PostOrderMappableContainer ()=default

  Distruttore virtuale di default.
- PostOrderMappableContainer & operator= (const PostOrderMappableContainer &)=delete
- PostOrderMappableContainer & operator= (PostOrderMappableContainer &&) noexcept=delete
- bool operator== (const PostOrderMappableContainer &) const noexcept=delete

  Operatore di confronto di uguaglianza disabilitato.
- bool operator!= (const PostOrderMappableContainer &) const noexcept=delete

  Operatore di confronto di disuguaglianza disabilitato.
- virtual void PostOrderMap (const MapFun fun)=0

  Applica una funzione a tutti gli elementi del contenitore in ordine PostOrder.
- void Map (const MapFun fun) override

  Applica la funzione mappante secondo la strategia PostOrder.

Public Member Functions inherited from lasd::MappableContainer< Data >

- virtual ~MappableContainer () noexcept=default

  Distruttore virtuale di default.
- MappableContainer & operator= (const MappableContainer &) noexcept=delete
- MappableContainer & operator= (MappableContainer &&) noexcept=delete
- bool operator== (const MappableContainer &) const noexcept=delete

  Operatore di confronto di uguaglianza disabilitato.
- bool operator!= (const MappableContainer &) const noexcept=delete

  Operatore di confronto di disuguaglianza disabilitato.

Public Member Functions inherited from lasd::TraversableContainer< Data >

- virtual ~TraversableContainer ()=default

  Distruttore virtuale di default.
- TraversableContainer & operator= (const TraversableContainer &)=delete
- TraversableContainer & operator= (TraversableContainer &&) noexcept=delete
- bool operator== (const TraversableContainer &) const noexcept=delete
- bool operator!= (const TraversableContainer &) const noexcept=delete
- template<typename Accumulator>
  Accumulator Fold (FoldFun< Accumulator > func, Accumulator base) const

  Esegue una riduzione (fold) sul contenitore.
- bool Exists (const Data &elem) const noexcept override

  Verifica se un elemento esiste nel contenitore.

Public Member Functions inherited from lasd::TestableContainer< Data >

- virtual ~TestableContainer ()=default

  Distruttore virtuale di default.
- TestableContainer & operator= (const TestableContainer &)=delete
- TestableContainer & operator= (TestableContainer &&) noexcept=delete
- bool operator== (const TestableContainer &) const noexcept=delete

  Operatore di confronto di uguaglianza disabilitato.
- bool operator!= (const TestableContainer &) const noexcept=delete

  Operatore di confronto di disuguaglianza disabilitato.

Public Member Functions inherited from lasd::Container

- virtual ~Container ()=default

  Destructor.
- Container & operator= (const Container &)=delete

  Copy assignment of abstract types is not possible.
- Container & operator= (Container &&) noexcept=delete

  Move assignment of abstract types is not possible.
- bool operator== (const Container &) const noexcept=delete
- bool operator!= (const Container &) const noexcept=delete
- virtual bool Empty () const noexcept

  La funzione Empty() controlla se la struttura è vuota (concrete function should not throw exceptions)
- virtual ulong Size () const noexcept

  La funzione Size() restituisce il numero di elementi presenti nella struttura (concrete function should not throw exceptions)

Public Member Functions inherited from lasd::PostOrderTraversableContainer< Data >

- virtual ~PostOrderTraversableContainer ()=default

  Distruttore virtuale di default.
- PostOrderTraversableContainer & operator= (const PostOrderTraversableContainer &)=delete
- PostOrderTraversableContainer & operator= (PostOrderTraversableContainer &&) noexcept=delete
- bool operator== (const PostOrderTraversableContainer &) const noexcept=delete
- bool operator!= (const PostOrderTraversableContainer &) const noexcept=delete
- virtual void PostOrderTraverse (TraverseFun func) const =0

  Esegue la traversata in post-ordine applicando una funzione a ogni elemento.
- template<typename Accumulator>
  Accumulator PostOrderFold (FoldFun< Accumulator > func, Accumulator base) const

  Esegue una riduzione in post-ordine.
- void Traverse (TraverseFun func) const override

  Implementazione della traversata base come traversata in post-ordine.

Additional Inherited Members

Public Types inherited from lasd::MappableContainer< Data >

- using MapFun = std::function<void(Data &)>

  Tipo di funzione mappante: accetta un riferimento modificabile a un elemento.

Public Types inherited from lasd::TraversableContainer< Data >

- using TraverseFun = std::function<void(const Data &)>

  Tipo funzione per la traversata (funzione unaria applicata a ogni dato).
- template<typename Accumulator>
  using FoldFun = std::function<Accumulator(const Data &, const Accumulator &)>

  Tipo funzione per fold (funzione binaria con accumulatore).

Public Types inherited from lasd::PostOrderTraversableContainer< Data >

- template<typename Accumulator>
  using FoldFun = typename TraversableContainer<Data>::FoldFun<Accumulator>

  Tipo funzione per fold in post-ordine.

Protected Member Functions inherited from lasd::TestableContainer< Data >

- TestableContainer ()=default

  Costruttore di default protetto.

Protected Member Functions inherited from lasd::Container

- Container ()=default

  Default constructor.

Protected Attributes inherited from lasd::Container

- ulong size = 0

6.10.1 Detailed Description

template<typename Data>
class lasd::PostOrderMappableContainer< Data >

Estensione di MappableContainer che specifica l'ordine PostOrder per la mappatura.

Template Parameters

Data	Tipo dei dati contenuti.

Definition at line 96 of file mappable.hpp.

6.10.2 Constructor & Destructor Documentation

6.10.2.1 ~PostOrderMappableContainer()

```
template<typename Data>
virtual lasd::PostOrderMappableContainer< Data >::~PostOrderMappableContainer ()  [virtual],
[default]
```

Distruttore virtuale di default.

6.10.3 Member Function Documentation

6.10.3.1 Map()

```
template<typename Data>
void lasd::PostOrderMappableContainer< Data >::Map (
            const MapFun fun)  [inline], [override], [virtual]
```

Applica la funzione mappante secondo la strategia PostOrder.

Parameters

fun	Funzione da applicare a ciascun elemento.

Implements lasd::MappableContainer< Data >.

Definition at line 12 of file mappable.cpp.

6.10.3.2 operator"!=()

```
template<typename Data>
bool lasd::PostOrderMappableContainer< Data >::operator!= (
            const PostOrderMappableContainer< Data > & ) const  [delete], [noexcept]
```

Operatore di confronto di disuguaglianza disabilitato.

6.10.3.3 operator=() [1/2]

```
template<typename Data>
PostOrderMappableContainer & lasd::PostOrderMappableContainer< Data >::operator= (
            const PostOrderMappableContainer< Data > & )  [delete]
```

6.10.3.4 operator=() [2/2]

```
template<typename Data>
PostOrderMappableContainer & lasd::PostOrderMappableContainer< Data >::operator= (
            PostOrderMappableContainer< Data > && )  [delete], [noexcept]
```

6.10.3.5 operator==()

```
template<typename Data>
bool lasd::PostOrderMappableContainer< Data >::operator== (
            const PostOrderMappableContainer< Data > & ) const  [delete], [noexcept]
```

Operatore di confronto di uguaglianza disabilitato.

6.10.3.6 PostOrderMap()

```
template<typename Data>
virtual void lasd::PostOrderMappableContainer< Data >::PostOrderMap (
            const MapFun fun)  [pure virtual]
```

Applica una funzione a tutti gli elementi del contenitore in ordine PostOrder.

Parameters

fun	Funzione che modifica ogni elemento tramite riferimento.

Implemented in lasd::LinearContainer< Data >, and lasd::MutableLinearContainer< Data >.

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

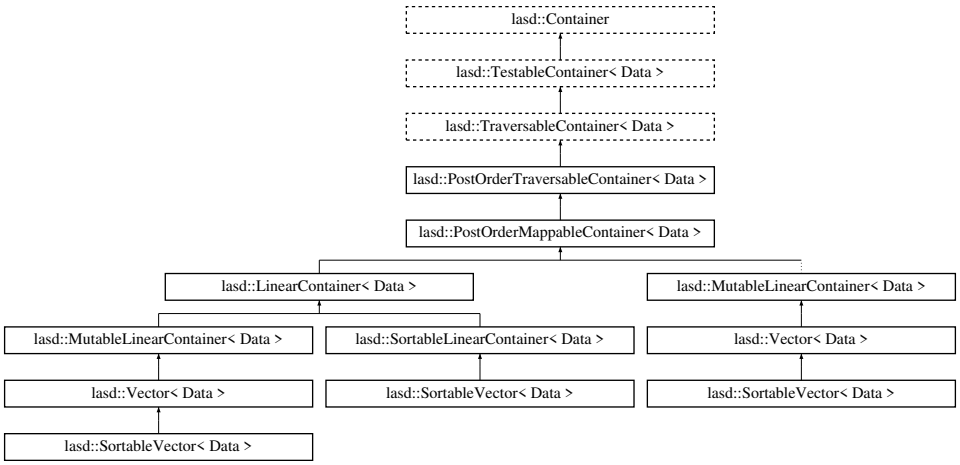
- Exercise1/container/mappable.hpp
- Exercise1/container/mappable.cpp

6.11 lasd::PostOrderTraversableContainer< Data > Class Template Reference

Classe astratta per contenitori con traversata in post-ordine.

`#include <traversable.hpp>`

Inheritance diagram for lasd::PostOrderTraversableContainer< Data >:



Public Types

- template<typename Accumulator>
  using FoldFun = typename TraversableContainer<Data>::FoldFun<Accumulator>

  Tipo funzione per fold in post-ordine.

Public Types inherited from lasd::TraversableContainer< Data >

- using TraverseFun = std::function<void(const Data &)>

  Tipo funzione per la traversata (funzione unaria applicata a ogni dato).
- template<typename Accumulator>
  using FoldFun = std::function<Accumulator(const Data &, const Accumulator &)>

  Tipo funzione per fold (funzione binaria con accumulatore).

Public Member Functions

- virtual ~PostOrderTraversableContainer ()=default

  Distruttore virtuale di default.
- PostOrderTraversableContainer & operator= (const PostOrderTraversableContainer &)=delete
- PostOrderTraversableContainer & operator= (PostOrderTraversableContainer &&) noexcept=delete
- bool operator== (const PostOrderTraversableContainer &) const noexcept=delete
- bool operator!= (const PostOrderTraversableContainer &) const noexcept=delete
- virtual void PostOrderTraverse (TraverseFun func) const =0

  Esegue la traversata in post-ordine applicando una funzione a ogni elemento.
- template<typename Accumulator>
  Accumulator PostOrderFold (FoldFun< Accumulator > func, Accumulator base) const

  Esegue una riduzione in post-ordine.
- void Traverse (TraverseFun func) const override

  Implementazione della traversata base come traversata in post-ordine.

Public Member Functions inherited from lasd::TraversableContainer< Data >

- virtual ~TraversableContainer ()=default

  Distruttore virtuale di default.
- TraversableContainer & operator= (const TraversableContainer &)=delete
- TraversableContainer & operator= (TraversableContainer &&) noexcept=delete
- bool operator== (const TraversableContainer &) const noexcept=delete
- bool operator!= (const TraversableContainer &) const noexcept=delete
- template<typename Accumulator>
  Accumulator Fold (FoldFun< Accumulator > func, Accumulator base) const

  Esegue una riduzione (fold) sul contenitore.
- bool Exists (const Data &elem) const noexcept override

  Verifica se un elemento esiste nel contenitore.

Public Member Functions inherited from lasd::TestableContainer< Data >

- virtual ~TestableContainer ()=default

  Distruttore virtuale di default.
- TestableContainer & operator= (const TestableContainer &)=delete
- TestableContainer & operator= (TestableContainer &&) noexcept=delete
- bool operator== (const TestableContainer &) const noexcept=delete

  Operatore di confronto di uguaglianza disabilitato.
- bool operator!= (const TestableContainer &) const noexcept=delete

  Operatore di confronto di disuguaglianza disabilitato.

Public Member Functions inherited from lasd::Container

- virtual ~Container ()=default

  Destructor.
- Container & operator= (const Container &)=delete

  Copy assignment of abstract types is not possible.
- Container & operator= (Container &&) noexcept=delete

  Move assignment of abstract types is not possible.
- bool operator== (const Container &) const noexcept=delete
- bool operator!= (const Container &) const noexcept=delete
- virtual bool Empty () const noexcept

  La funzione Empty() controlla se la struttura è vuota (concrete function should not throw exceptions)
- virtual ulong Size () const noexcept

  La funzione Size() restituisce il numero di elementi presenti nella struttura (concrete function should not throw exceptions)

Additional Inherited Members

Protected Member Functions inherited from lasd::TestableContainer< Data >

- TestableContainer ()=default

  Costruttore di default protetto.

Protected Member Functions inherited from lasd::Container

- Container ()=default

  Default constructor.

Protected Attributes inherited from lasd::Container

- ulong size = 0

6.11.1 Detailed Description

template<typename Data>
class lasd::PostOrderTraversableContainer< Data >

Classe astratta per contenitori con traversata in post-ordine.

Template Parameters

Data	Tipo dei dati contenuti.

Definition at line 98 of file traversable.hpp.

6.11.2 Member Typedef Documentation

6.11.2.1 FoldFun

```
template<typename Data>
template<typename Accumulator>
using lasd::PostOrderTraversableContainer< Data >::FoldFun = typename TraversableContainer<Data>↩
::FoldFun<Accumulator>
```

Tipo funzione per fold in post-ordine.

Definition at line 121 of file traversable.hpp.

6.11.3 Constructor & Destructor Documentation

6.11.3.1 ~PostOrderTraversableContainer()

```
template<typename Data>
virtual lasd::PostOrderTraversableContainer< Data >::~PostOrderTraversableContainer ()  [virtual],
[default]
```

Distruttore virtuale di default.

6.11.4 Member Function Documentation

6.11.4.1 operator"!=()

```
template<typename Data>
bool lasd::PostOrderTraversableContainer< Data >::operator!= (
            const PostOrderTraversableContainer< Data > & ) const  [delete], [noexcept]
```

6.11.4.2 operator=() [1/2]

```
template<typename Data>
PostOrderTraversableContainer & lasd::PostOrderTraversableContainer< Data >::operator= (
            const PostOrderTraversableContainer< Data > & )  [delete]
```

6.11.4.3 operator=() [2/2]

```
template<typename Data>
PostOrderTraversableContainer & lasd::PostOrderTraversableContainer< Data >::operator= (
            PostOrderTraversableContainer< Data > && )  [delete], [noexcept]
```

6.11.4.4 operator==()

```
template<typename Data>
bool lasd::PostOrderTraversableContainer< Data >::operator== (
            const PostOrderTraversableContainer< Data > & ) const  [delete], [noexcept]
```

6.11.4.5 PostOrderFold()

```
template<typename Data>
template<typename Accumulator>
Accumulator lasd::PostOrderTraversableContainer< Data >::PostOrderFold (
            FoldFun< Accumulator > func,
            Accumulator base) const  [inline]
```

Esegue una riduzione in post-ordine.

Template Parameters

Accumulator	Tipo dell'accumulatore.

Parameters

func	Funzione di riduzione.
base	Valore iniziale dell'accumulatore.

Returns

Il risultato finale del fold.

Definition at line 55 of file traversable.cpp.

6.11.4.6 PostOrderTraverse()

```
template<typename Data>
virtual void lasd::PostOrderTraversableContainer< Data >::PostOrderTraverse (
            TraverseFun func) const  [pure virtual]
```

Esegue la traversata in post-ordine applicando una funzione a ogni elemento.

Parameters

func	Funzione da applicare agli elementi.

Implemented in lasd::LinearContainer< Data >.

6.11.4.7 Traverse()

```
template<typename Data>
void lasd::PostOrderTraversableContainer< Data >::Traverse (
            TraverseFun func) const  [inline], [override], [virtual]
```

Implementazione della traversata base come traversata in post-ordine.

Parameters

func	Funzione da applicare agli elementi.

Implements lasd::TraversableContainer< Data >.

Definition at line 48 of file traversable.cpp.

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

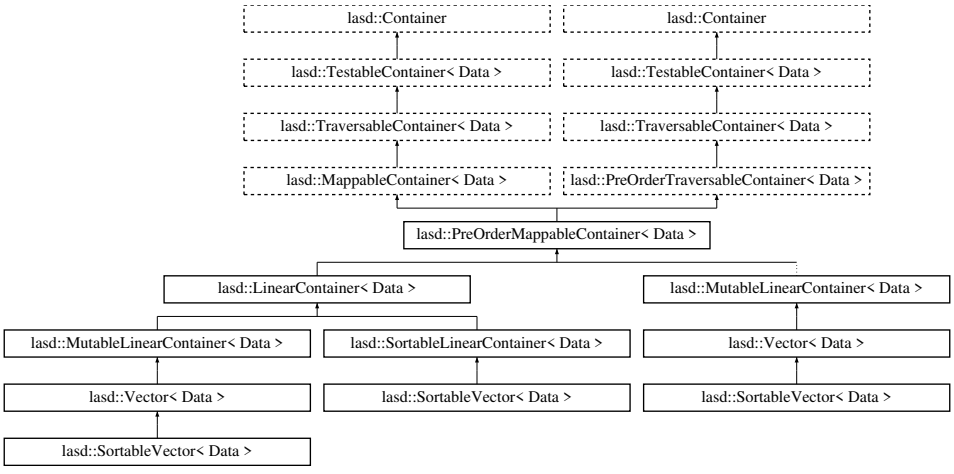
- Exercise1/container/traversable.hpp
- Exercise1/container/traversable.cpp

6.12 lasd::PreOrderMappableContainer< Data > Class Template Reference

Estensione di MappableContainer che specifica l'ordine PreOrder per la mappatura.

`#include <mappable.hpp>`

Inheritance diagram for lasd::PreOrderMappableContainer< Data >:



Public Member Functions

- virtual ~PreOrderMappableContainer ()=default

  Distruttore virtuale di default.
- PreOrderMappableContainer & operator= (const PreOrderMappableContainer &)=delete
- PreOrderMappableContainer & operator= (PreOrderMappableContainer &&) noexcept=delete
- bool operator== (const PreOrderMappableContainer &) const noexcept=delete

  Operatore di confronto di uguaglianza disabilitato.
- bool operator!= (const PreOrderMappableContainer &) const noexcept=delete

  Operatore di confronto di disuguaglianza disabilitato.
- virtual void PreOrderMap (MapFun fun)=0

  Applica una funzione a tutti gli elementi del contenitore in ordine PreOrder.
- void Map (MapFun fun) override

  Applica la funzione mappante secondo la strategia PreOrder.

Public Member Functions inherited from lasd::MappableContainer< Data >

- virtual ~MappableContainer () noexcept=default

  Distruttore virtuale di default.
- MappableContainer & operator= (const MappableContainer &) noexcept=delete
- MappableContainer & operator= (MappableContainer &&) noexcept=delete
- bool operator== (const MappableContainer &) const noexcept=delete

  Operatore di confronto di uguaglianza disabilitato.
- bool operator!= (const MappableContainer &) const noexcept=delete

  Operatore di confronto di disuguaglianza disabilitato.

Public Member Functions inherited from lasd::TraversableContainer< Data >

- virtual ~TraversableContainer ()=default

  Distruttore virtuale di default.
- TraversableContainer & operator= (const TraversableContainer &)=delete
- TraversableContainer & operator= (TraversableContainer &&) noexcept=delete
- bool operator== (const TraversableContainer &) const noexcept=delete
- bool operator!= (const TraversableContainer &) const noexcept=delete
- template<typename Accumulator>
  Accumulator Fold (FoldFun< Accumulator > func, Accumulator base) const

  Esegue una riduzione (fold) sul contenitore.
- bool Exists (const Data &elem) const noexcept override

  Verifica se un elemento esiste nel contenitore.

Public Member Functions inherited from lasd::TestableContainer< Data >

- virtual ~TestableContainer ()=default

  Distruttore virtuale di default.
- TestableContainer & operator= (const TestableContainer &)=delete
- TestableContainer & operator= (TestableContainer &&) noexcept=delete
- bool operator== (const TestableContainer &) const noexcept=delete

  Operatore di confronto di uguaglianza disabilitato.
- bool operator!= (const TestableContainer &) const noexcept=delete

  Operatore di confronto di disuguaglianza disabilitato.

Public Member Functions inherited from lasd::Container

- virtual ~Container ()=default

  Destructor.
- Container & operator= (const Container &)=delete

  Copy assignment of abstract types is not possible.
- Container & operator= (Container &&) noexcept=delete

  Move assignment of abstract types is not possible.
- bool operator== (const Container &) const noexcept=delete
- bool operator!= (const Container &) const noexcept=delete
- virtual bool Empty () const noexcept

  La funzione Empty() controlla se la struttura è vuota (concrete function should not throw exceptions)
- virtual ulong Size () const noexcept

  La funzione Size() restituisce il numero di elementi presenti nella struttura (concrete function should not throw exceptions)

Public Member Functions inherited from lasd::PreOrderTraversableContainer< Data >

- virtual ~PreOrderTraversableContainer ()=default

  Distruttore virtuale di default.
- PreOrderTraversableContainer & operator= (const PreOrderTraversableContainer &)=delete
- PreOrderTraversableContainer & operator= (PreOrderTraversableContainer &&) noexcept=delete
- bool operator== (const PreOrderTraversableContainer &) const noexcept=delete
- bool operator!= (const PreOrderTraversableContainer &) const noexcept=delete
- virtual void PreOrderTraverse (const TraverseFun func) const =0

  Esegue la traversata in pre-ordine applicando una funzione a ogni elemento.
- template<typename Accumulator>
  Accumulator PreOrderFold (FoldFun< Accumulator > func, Accumulator base) const

  Esegue una riduzione in pre-ordine.
- void Traverse (TraverseFun func) const override

  Implementazione della traversata base come traversata in pre-ordine.

Additional Inherited Members

Public Types inherited from lasd::MappableContainer< Data >

- using MapFun = std::function<void(Data &)>

  Tipo di funzione mappante: accetta un riferimento modificabile a un elemento.

Public Types inherited from lasd::TraversableContainer< Data >

- using TraverseFun = std::function<void(const Data &)>

  Tipo funzione per la traversata (funzione unaria applicata a ogni dato).
- template<typename Accumulator>
  using FoldFun = std::function<Accumulator(const Data &, const Accumulator &)>

  Tipo funzione per fold (funzione binaria con accumulatore).

Public Types inherited from lasd::PreOrderTraversableContainer< Data >

- template<typename Accumulator>
  using FoldFun = typename TraversableContainer<Data>::FoldFun<Accumulator>

  Tipo funzione per fold in pre-ordine.

Protected Member Functions inherited from lasd::TestableContainer< Data >

- TestableContainer ()=default

  Costruttore di default protetto.

Protected Member Functions inherited from lasd::Container

- Container ()=default

  Default constructor.

Protected Attributes inherited from lasd::Container

- ulong size = 0

6.12.1 Detailed Description

template<typename Data>
class lasd::PreOrderMappableContainer< Data >

Estensione di MappableContainer che specifica l'ordine PreOrder per la mappatura.

Template Parameters

Data	Tipo dei dati contenuti.

Definition at line 54 of file mappable.hpp.

6.12.2 Constructor & Destructor Documentation

6.12.2.1 ~PreOrderMappableContainer()

```
template<typename Data>
virtual lasd::PreOrderMappableContainer< Data >::~PreOrderMappableContainer ()  [virtual],
[default]
```

Distruttore virtuale di default.

6.12.3 Member Function Documentation

6.12.3.1 Map()

```
template<typename Data>
void lasd::PreOrderMappableContainer< Data >::Map (
            MapFun fun)  [inline], [override], [virtual]
```

Applica la funzione mappante secondo la strategia PreOrder.

Parameters

fun	Funzione da applicare a ciascun elemento.

Implements lasd::MappableContainer< Data >.

Definition at line 6 of file mappable.cpp.

6.12.3.2 operator"!=()

```
template<typename Data>
bool lasd::PreOrderMappableContainer< Data >::operator!= (
            const PreOrderMappableContainer< Data > & ) const  [delete], [noexcept]
```

Operatore di confronto di disuguaglianza disabilitato.

6.12.3.3 operator=() [1/2]

```
template<typename Data>
PreOrderMappableContainer & lasd::PreOrderMappableContainer< Data >::operator= (
            const PreOrderMappableContainer< Data > & )  [delete]
```

6.12.3.4 operator=() [2/2]

```
template<typename Data>
PreOrderMappableContainer & lasd::PreOrderMappableContainer< Data >::operator= (
            PreOrderMappableContainer< Data > && )  [delete], [noexcept]
```

6.12.3.5 operator==()

```
template<typename Data>
bool lasd::PreOrderMappableContainer< Data >::operator== (
            const PreOrderMappableContainer< Data > & ) const  [delete], [noexcept]
```

Operatore di confronto di uguaglianza disabilitato.

6.12.3.6 PreOrderMap()

```
template<typename Data>
virtual void lasd::PreOrderMappableContainer< Data >::PreOrderMap (
            MapFun fun)  [pure virtual]
```

Applica una funzione a tutti gli elementi del contenitore in ordine PreOrder.

Parameters

fun	Funzione che modifica ogni elemento tramite riferimento.

Implemented in lasd::LinearContainer< Data >, and lasd::MutableLinearContainer< Data >.

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

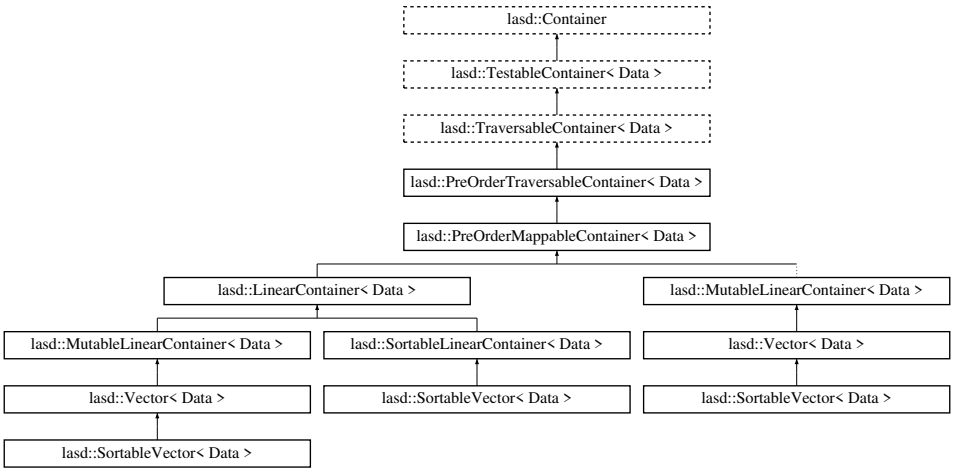
- Exercise1/container/mappable.hpp
- Exercise1/container/mappable.cpp

6.13 lasd::PreOrderTraversableContainer< Data > Class Template Reference

Classe astratta per contenitori con traversata in pre-ordine.

`#include <traversable.hpp>`

Inheritance diagram for lasd::PreOrderTraversableContainer< Data >:



Public Types

- template<typename Accumulator>
  using FoldFun = typename TraversableContainer<Data>::FoldFun<Accumulator>

  Tipo funzione per fold in pre-ordine.

Public Types inherited from lasd::TraversableContainer< Data >

- using TraverseFun = std::function<void(const Data &)>

  Tipo funzione per la traversata (funzione unaria applicata a ogni dato).
- template<typename Accumulator>
  using FoldFun = std::function<Accumulator(const Data &, const Accumulator &)>

  Tipo funzione per fold (funzione binaria con accumulatore).

Public Member Functions

- virtual ~PreOrderTraversableContainer ()=default

  Distruttore virtuale di default.
- PreOrderTraversableContainer & operator= (const PreOrderTraversableContainer &)=delete
- PreOrderTraversableContainer & operator= (PreOrderTraversableContainer &&) noexcept=delete
- bool operator== (const PreOrderTraversableContainer &) const noexcept=delete
- bool operator!= (const PreOrderTraversableContainer &) const noexcept=delete
- virtual void PreOrderTraverse (const TraverseFun func) const =0

  Esegue la traversata in pre-ordine applicando una funzione a ogni elemento.
- template<typename Accumulator>
  Accumulator PreOrderFold (FoldFun< Accumulator > func, Accumulator base) const

  Esegue una riduzione in pre-ordine.
- void Traverse (TraverseFun func) const override

  Implementazione della traversata base come traversata in pre-ordine.

Public Member Functions inherited from lasd::TraversableContainer< Data >

- virtual ~TraversableContainer ()=default

  Distruttore virtuale di default.
- TraversableContainer & operator= (const TraversableContainer &)=delete
- TraversableContainer & operator= (TraversableContainer &&) noexcept=delete
- bool operator== (const TraversableContainer &) const noexcept=delete
- bool operator!= (const TraversableContainer &) const noexcept=delete
- template<typename Accumulator>
  Accumulator Fold (FoldFun< Accumulator > func, Accumulator base) const

  Esegue una riduzione (fold) sul contenitore.
- bool Exists (const Data &elem) const noexcept override

  Verifica se un elemento esiste nel contenitore.

Public Member Functions inherited from lasd::TestableContainer< Data >

- virtual ~TestableContainer ()=default

  Distruttore virtuale di default.
- TestableContainer & operator= (const TestableContainer &)=delete
- TestableContainer & operator= (TestableContainer &&) noexcept=delete
- bool operator== (const TestableContainer &) const noexcept=delete

  Operatore di confronto di uguaglianza disabilitato.
- bool operator!= (const TestableContainer &) const noexcept=delete

  Operatore di confronto di disuguaglianza disabilitato.

Public Member Functions inherited from lasd::Container

- virtual ~Container ()=default

  Destructor.
- Container & operator= (const Container &)=delete

  Copy assignment of abstract types is not possible.
- Container & operator= (Container &&) noexcept=delete

  Move assignment of abstract types is not possible.
- bool operator== (const Container &) const noexcept=delete
- bool operator!= (const Container &) const noexcept=delete
- virtual bool Empty () const noexcept

  La funzione Empty() controlla se la struttura è vuota (concrete function should not throw exceptions)
- virtual ulong Size () const noexcept

  La funzione Size() restituisce il numero di elementi presenti nella struttura (concrete function should not throw exceptions)

Additional Inherited Members

Protected Member Functions inherited from lasd::TestableContainer< Data >

- TestableContainer ()=default

  Costruttore di default protetto.

Protected Member Functions inherited from lasd::Container

- Container ()=default

  Default constructor.

Protected Attributes inherited from lasd::Container

- ulong size = 0

6.13.1 Detailed Description

template<typename Data>
class lasd::PreOrderTraversableContainer< Data >

Classe astratta per contenitori con traversata in pre-ordine.

Template Parameters

Data	Tipo dei dati contenuti.

Definition at line 56 of file traversable.hpp.

6.13.2 Member Typedef Documentation

6.13.2.1 FoldFun

```
template<typename Data>
template<typename Accumulator>
using lasd::PreOrderTraversableContainer< Data >::FoldFun = typename TraversableContainer<Data>↩
::FoldFun<Accumulator>
```

Tipo funzione per fold in pre-ordine.

Definition at line 80 of file traversable.hpp.

6.13.3 Constructor & Destructor Documentation

6.13.3.1 ~PreOrderTraversableContainer()

```
template<typename Data>
virtual lasd::PreOrderTraversableContainer< Data >::~PreOrderTraversableContainer ()  [virtual],
[default]
```

Distruttore virtuale di default.

6.13.4 Member Function Documentation

6.13.4.1 operator"!=()

```
template<typename Data>
bool lasd::PreOrderTraversableContainer< Data >::operator!= (
            const PreOrderTraversableContainer< Data > & ) const  [delete], [noexcept]
```

6.13.4.2 operator=() [1/2]

```
template<typename Data>
PreOrderTraversableContainer & lasd::PreOrderTraversableContainer< Data >::operator= (
            const PreOrderTraversableContainer< Data > & )  [delete]
```

6.13.4.3 operator=() [2/2]

```
template<typename Data>
PreOrderTraversableContainer & lasd::PreOrderTraversableContainer< Data >::operator= (
            PreOrderTraversableContainer< Data > && )  [delete], [noexcept]
```

6.13.4.4 operator==()

```
template<typename Data>
bool lasd::PreOrderTraversableContainer< Data >::operator== (
            const PreOrderTraversableContainer< Data > & ) const  [delete], [noexcept]
```

6.13.4.5 PreOrderFold()

```
template<typename Data>
template<typename Accumulator>
Accumulator lasd::PreOrderTraversableContainer< Data >::PreOrderFold (
            FoldFun< Accumulator > func,
            Accumulator base) const  [inline]
```

Esegue una riduzione in pre-ordine.

Template Parameters

Accumulator	Tipo dell'accumulatore.

Parameters

func	Funzione di riduzione.
base	Valore iniziale dell'accumulatore.

Returns

Il risultato finale del fold.

Definition at line 36 of file traversable.cpp.

6.13.4.6 PreOrderTraverse()

```
template<typename Data>
virtual void lasd::PreOrderTraversableContainer< Data >::PreOrderTraverse (
            const TraverseFun func) const  [pure virtual]
```

Esegue la traversata in pre-ordine applicando una funzione a ogni elemento.

Parameters

func	Funzione da applicare agli elementi.

Implemented in lasd::LinearContainer< Data >.

6.13.4.7 Traverse()

```
template<typename Data>
void lasd::PreOrderTraversableContainer< Data >::Traverse (
            TraverseFun func) const  [inline], [override], [virtual]
```

Implementazione della traversata base come traversata in pre-ordine.

Parameters

func	Funzione da applicare agli elementi.

Implements lasd::TraversableContainer< Data >.

Definition at line 29 of file traversable.cpp.

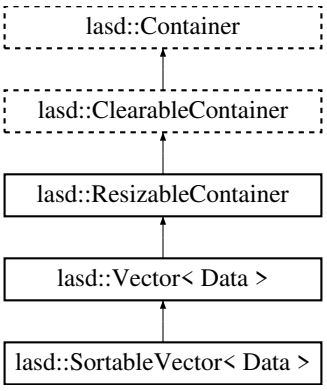
The documentation for this class was generated from the following files:

- Exercise1/container/traversable.hpp
- Exercise1/container/traversable.cpp

6.14 lasd::ResizableContainer Class Reference

```
#include <container.hpp>
```

Inheritance diagram for lasd::ResizableContainer:



Public Member Functions

- virtual ~ResizableContainer ()=default

  Destructor.
- ResizableContainer & operator= (const ResizableContainer &Ccon) noexcept=delete

  Copy assignment.
- ResizableContainer & operator= (ResizableContainer &&Ccon) noexcept=delete

  Move assignment.
- bool operator== (const ResizableContainer &Ccon) const noexcept=delete
- bool operator!= (const ResizableContainer &Ccon) const noexcept=delete
- virtual void Resize (ulong)=0

  Resize è metodo virtuale puro.
- void Clear () override

  Anche se non è possibile dichiarare oggetti di una classe astratta, possiamo usare i metodi virtuali puri che questa fornisce per implementare delle funzionalità di base. Sarà poi il polimorfismo ad eseguire un dispatching dinamico per chiamare i metodi delle classi concrete che estendono la classe astratta. Ad esempio, a questo livello di astrazione possiamo già fornire una implementazione di base per il metodo Clear anche senza sapere come svuotare una struttura dati generica. Questa implementazione prevede che la struttura dati venga ridimensionata a 0.

Public Member Functions inherited from lasd::ClearableContainer

- virtual ~ClearableContainer ()=default

  Destructor.
- ClearableContainer & operator= (const ClearableContainer &)=delete

  Copy assignment.
- ClearableContainer & operator= (ClearableContainer &&) noexcept=delete

  Move assignment.
- bool operator== (const ClearableContainer &Ccon) const noexcept=delete

  Comparison operators.
- bool operator!= (const ClearableContainer &Ccon) const noexcept=delete

Public Member Functions inherited from lasd::Container

- virtual ~Container ()=default

  Destructor.
- Container & operator= (const Container &)=delete

  Copy assignment of abstract types is not possible.
- Container & operator= (Container &&) noexcept=delete

  Move assignment of abstract types is not possible.
- bool operator== (const Container &) const noexcept=delete
- bool operator!= (const Container &) const noexcept=delete
- virtual bool Empty () const noexcept

  La funzione Empty() controlla se la struttura è vuota (concrete function should not throw exceptions)
- virtual ulong Size () const noexcept

  La funzione Size() restituisce il numero di elementi presenti nella struttura (concrete function should not throw exceptions)

Additional Inherited Members

Protected Member Functions inherited from lasd::Container

- Container ()=default

  Default constructor.

Protected Attributes inherited from lasd::Container

- ulong size = 0

6.14.1 Detailed Description

ResizableContainer rappresenta un'estensione della classe Clearable e funge da collettore per tutte quelle strutture dati che possono essere ridimensionate, ovvero che possono modificare la propria dimensione in maniera dinamica.

Definition at line 71 of file container.hpp.

6.14.2 Constructor & Destructor Documentation

6.14.2.1 ~ResizableContainer()

```
virtual lasd::ResizableContainer::~ResizableContainer ()  [virtual], [default]
```

Destructor.

6.14.3 Member Function Documentation

6.14.3.1 Clear()

```
void lasd::ResizableContainer::Clear ()  [inline], [override], [virtual]
```

Anche se non è possibile dichiarare oggetti di una classe astratta, possiamo usare i metodi virtuali puri che questa fornisce per implementare delle funzionalità di base. Sarà poi il polimorfismo ad eseguire un dispatching dinamico per chiamare i metodi delle classi concrete che estendono la classe astratta. Ad esempio, a questo livello di astrazione possiamo già fornire una implementazione di base per il metodo Clear anche senza sapere come svuotare una struttura dati generica. Questa implementazione prevede che la struttura dati venga ridimensionata a 0.

Implements lasd::ClearableContainer.

Reimplemented in lasd::Vector< Data >.

Definition at line 97 of file container.hpp.

6.14.3.2 operator"!=()

```
bool lasd::ResizableContainer::operator!= (
            const ResizableContainer & Ccon) const  [delete], [noexcept]
```

6.14.3.3 operator=() [1/2]

```
ResizableContainer & lasd::ResizableContainer::operator= (
            const ResizableContainer & Ccon)  [delete], [noexcept]
```

Copy assignment.

6.14.3.4 operator=() [2/2]

```
ResizableContainer & lasd::ResizableContainer::operator= (
            ResizableContainer && Ccon)  [delete], [noexcept]
```

Move assignment.

6.14.3.5 operator==()

```
bool lasd::ResizableContainer::operator== (
            const ResizableContainer & Ccon) const  [delete], [noexcept]
```

6.14.3.6 Resize()

```
virtual void lasd::ResizableContainer::Resize (
            ulong )  [pure virtual]
```

Resize è metodo virtuale puro.

Parameters

ulong	un argomento di tipo intero per specificare la nuova dimensione del container

Implemented in lasd::Vector< Data >.

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

- Exercise1/container/container.hpp

6.15 lasd::Set< Data > Class Template Reference

```
#include <set.hpp>
```

6.15.1 Detailed Description

template<typename Data>
class lasd::Set< Data >

Definition at line 17 of file set.hpp.

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

- Exercise1/set/set.hpp

6.16 lasd::SetLst< Data > Class Template Reference

`#include <setlst.hpp>`

6.16.1 Detailed Description

template<typename Data>
class lasd::SetLst< Data >

Definition at line 17 of file setlst.hpp.

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

- Exercise1/set/lst/setlst.hpp

6.17 lasd::SetVec< Data > Class Template Reference

`#include <setvec.hpp>`

6.17.1 Detailed Description

template<typename Data>
class lasd::SetVec< Data >

Definition at line 17 of file setvec.hpp.

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

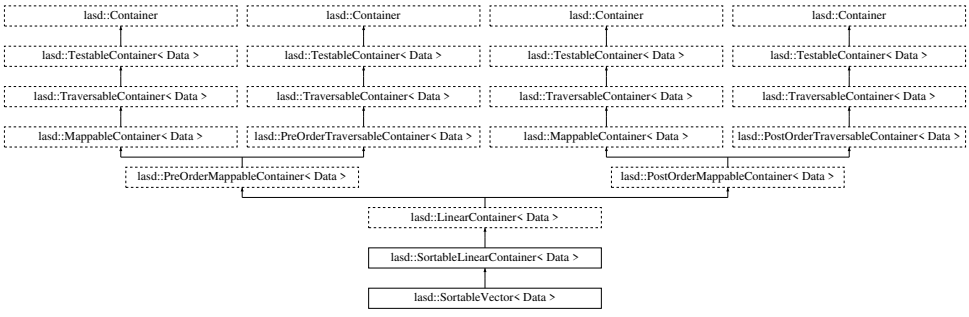
- Exercise1/set/vec/setvec.hpp

6.18 lasd::SortableLinearContainer< Data > Class Template Reference

Classe astratta che rappresenta un contenitore lineare ordinabile.

`#include <linear.hpp>`

Inheritance diagram for lasd::SortableLinearContainer< Data >:



Public Member Functions

- virtual ~SortableLinearContainer () noexcept=default

  Distruttore virtuale di default.
- SortableLinearContainer & operator= (const SortableLinearContainer &) noexcept=delete
- SortableLinearContainer & operator= (SortableLinearContainer &&) noexcept=delete
- bool operator== (const SortableLinearContainer &) const noexcept

  Operatore di confronto di uguaglianza.
- bool operator!= (const SortableLinearContainer &) const noexcept

  Operatore di confronto di disuguaglianza.
- void Sort () noexcept

  Ordina il contenitore secondo un algoritmo non specificato (QuickSort/Insertsort).

Public Member Functions inherited from lasd::LinearContainer< Data >

- virtual ~LinearContainer ()=default

  Distruttore virtuale di default.
- LinearContainer & operator= (const LinearContainer &)=delete
- LinearContainer & operator= (LinearContainer &&) noexcept=delete
- bool operator== (const LinearContainer &) const noexcept

  Operatore di confronto di uguaglianza.
- bool operator!= (const LinearContainer &) const noexcept

  Operatore di confronto di disuguaglianza.
- virtual const Data & operator[ ] (const ulong int index) const =0

  Accesso in sola lettura all'elemento in posizione specifica.
- virtual Data & operator[ ] (const ulong int index)=0

  Accesso in lettura/scrittura all'elemento in posizione specifica.
- virtual const Data & Front () const

  Restituisce il primo elemento (costante).
- virtual Data & Front ()

  Restituisce il primo elemento.
- virtual const Data & Back () const

  Restituisce l'ultimo elemento (costante).
- virtual Data & Back ()

  Restituisce l'ultimo elemento.
- void Traverse (TraverseFun) const override

  Traversamento in ordine predefinito.
- void PreOrderTraverse (TraverseFun) const override

  Traversamento in ordine PreOrder.
- void PostOrderTraverse (TraverseFun) const override

  Traversamento in ordine PostOrder.
- void Map (MapFun) override

  Mappatura in ordine predefinito.
- void PreOrderMap (MapFun) override

  Mappatura in ordine PreOrder.
- void PostOrderMap (MapFun) override

  Mappatura in ordine PostOrder.

Public Member Functions inherited from lasd::PreOrderMappableContainer< Data >

- virtual ~PreOrderMappableContainer ()=default

  Distruttore virtuale di default.
- PreOrderMappableContainer & operator= (const PreOrderMappableContainer &)=delete
- PreOrderMappableContainer & operator= (PreOrderMappableContainer &&) noexcept=delete
- bool operator== (const PreOrderMappableContainer &) const noexcept=delete

  Operatore di confronto di uguaglianza disabilitato.
- bool operator!= (const PreOrderMappableContainer &) const noexcept=delete

  Operatore di confronto di disuguaglianza disabilitato.
- void Map (MapFun fun) override

  Applica la funzione mappante secondo la strategia PreOrder.

Public Member Functions inherited from lasd::MappableContainer< Data >

- virtual ~MappableContainer () noexcept=default

  Distruttore virtuale di default.
- MappableContainer & operator= (const MappableContainer &) noexcept=delete
- MappableContainer & operator= (MappableContainer &&) noexcept=delete
- bool operator== (const MappableContainer &) const noexcept=delete

  Operatore di confronto di uguaglianza disabilitato.
- bool operator!= (const MappableContainer &) const noexcept=delete

  Operatore di confronto di disuguaglianza disabilitato.

Public Member Functions inherited from lasd::TraversableContainer< Data >

- virtual ~TraversableContainer ()=default

  Distruttore virtuale di default.
- TraversableContainer & operator= (const TraversableContainer &)=delete
- TraversableContainer & operator= (TraversableContainer &&) noexcept=delete
- bool operator== (const TraversableContainer &) const noexcept=delete
- bool operator!= (const TraversableContainer &) const noexcept=delete
- template<typename Accumulator>
  Accumulator Fold (FoldFun< Accumulator > func, Accumulator base) const

  Esegue una riduzione (fold) sul contenitore.
- bool Exists (const Data &elem) const noexcept override

  Verifica se un elemento esiste nel contenitore.

Public Member Functions inherited from lasd::TestableContainer< Data >

- virtual ~TestableContainer ()=default

  Distruttore virtuale di default.
- TestableContainer & operator= (const TestableContainer &)=delete
- TestableContainer & operator= (TestableContainer &&) noexcept=delete
- bool operator== (const TestableContainer &) const noexcept=delete

  Operatore di confronto di uguaglianza disabilitato.
- bool operator!= (const TestableContainer &) const noexcept=delete

  Operatore di confronto di disuguaglianza disabilitato.

Public Member Functions inherited from lasd::Container

- virtual ~Container ()=default

  Destructor.
- Container & operator= (const Container &)=delete

  Copy assignment of abstract types is not possible.
- Container & operator= (Container &&) noexcept=delete

  Move assignment of abstract types is not possible.
- bool operator== (const Container &) const noexcept=delete
- bool operator!= (const Container &) const noexcept=delete
- virtual bool Empty () const noexcept

  La funzione Empty() controlla se la struttura è vuota (concrete function should not throw exceptions)
- virtual ulong Size () const noexcept

  La funzione Size() restituisce il numero di elementi presenti nella struttura (concrete function should not throw exceptions)

Public Member Functions inherited from lasd::PreOrderTraversableContainer< Data >

- virtual ~PreOrderTraversableContainer ()=default

  Distruttore virtuale di default.
- PreOrderTraversableContainer & operator= (const PreOrderTraversableContainer &)=delete
- PreOrderTraversableContainer & operator= (PreOrderTraversableContainer &&) noexcept=delete
- bool operator== (const PreOrderTraversableContainer &) const noexcept=delete
- bool operator!= (const PreOrderTraversableContainer &) const noexcept=delete
- template<typename Accumulator>
  Accumulator PreOrderFold (FoldFun< Accumulator > func, Accumulator base) const

  Esegue una riduzione in pre-ordine.
- void Traverse (TraverseFun func) const override

  Implementazione della traversata base come traversata in pre-ordine.

Public Member Functions inherited from lasd::PostOrderMappableContainer< Data >

- virtual ~PostOrderMappableContainer ()=default

  Distruttore virtuale di default.
- PostOrderMappableContainer & operator= (const PostOrderMappableContainer &)=delete
- PostOrderMappableContainer & operator= (PostOrderMappableContainer &&) noexcept=delete
- bool operator== (const PostOrderMappableContainer &) const noexcept=delete

  Operatore di confronto di uguaglianza disabilitato.
- bool operator!= (const PostOrderMappableContainer &) const noexcept=delete

  Operatore di confronto di disuguaglianza disabilitato.
- void Map (const MapFun fun) override

  Applica la funzione mappante secondo la strategia PostOrder.

Public Member Functions inherited from lasd::PostOrderTraversableContainer< Data >

- virtual ~PostOrderTraversableContainer ()=default

  Distruttore virtuale di default.
- PostOrderTraversableContainer & operator= (const PostOrderTraversableContainer &)=delete
- PostOrderTraversableContainer & operator= (PostOrderTraversableContainer &&) noexcept=delete
- bool operator== (const PostOrderTraversableContainer &) const noexcept=delete
- bool operator!= (const PostOrderTraversableContainer &) const noexcept=delete
- template<typename Accumulator>
  Accumulator PostOrderFold (FoldFun< Accumulator > func, Accumulator base) const

  Esegue una riduzione in post-ordine.
- void Traverse (TraverseFun func) const override

  Implementazione della traversata base come traversata in post-ordine.

Protected Member Functions

- void quickSort (ulong left, ulong right) noexcept
  Esegue QuickSort ricorsivamente tra due indici.
- ulong partition (ulong left, ulong right) noexcept
  Esegue la partizione per QuickSort.

Protected Member Functions inherited from lasd::TestableContainer< Data >

- TestableContainer ()=default
  Costruttore di default protetto.

Protected Member Functions inherited from lasd::Container

- Container ()=default
  Default constructor.

Protected Attributes

- ulong size

Protected Attributes inherited from lasd::LinearContainer< Data >

- ulong size

Protected Attributes inherited from lasd::Container

- ulong size = 0

Additional Inherited Members

Public Types inherited from lasd::MappableContainer< Data >

- using MapFun = std::function<void(Data &)>
  Tipo di funzione mappante: accetta un riferimento modificabile a un elemento.

Public Types inherited from lasd::TraversableContainer< Data >

- using TraverseFun = std::function<void(const Data &)>
  Tipo funzione per la traversata (funzione unaria applicata a ogni dato).
- template<typename Accumulator>
  using FoldFun = std::function<Accumulator(const Data &, const Accumulator &)>
  Tipo funzione per fold (funzione binaria con accumulatore).

Public Types inherited from lasd::PreOrderTraversableContainer< Data >

- template<typename Accumulator>
  using FoldFun = typename TraversableContainer<Data>::FoldFun<Accumulator>

  Tipo funzione per fold in pre-ordine.

Public Types inherited from lasd::PostOrderTraversableContainer< Data >

- template<typename Accumulator>
  using FoldFun = typename TraversableContainer<Data>::FoldFun<Accumulator>

  Tipo funzione per fold in post-ordine.

6.18.1 Detailed Description

template<typename Data>
class lasd::SortableLinearContainer< Data >

Classe astratta che rappresenta un contenitore lineare ordinabile.

Estende LinearContainer aggiungendo funzionalità di ordinamento.

Template Parameters

Data	Tipo dei dati contenuti.

Definition at line 177 of file linear.hpp.

6.18.2 Constructor & Destructor Documentation

6.18.2.1 ~SortableLinearContainer()

```
template<typename Data>
virtual lasd::SortableLinearContainer< Data >::~SortableLinearContainer ()  [virtual], [default],
[noexcept]
```

Distruttore virtuale di default.

6.18.3 Member Function Documentation

6.18.3.1 operator"!=()

```
template<typename Data>
bool lasd::SortableLinearContainer< Data >::operator!= (
            const SortableLinearContainer< Data > & con) const  [inline], [noexcept]
```

Operatore di confronto di disuguaglianza.

Definition at line 131 of file linear.cpp.

6.18.3.2 operator=() [1/2]

```
template<typename Data>
SortableLinearContainer & lasd::SortableLinearContainer< Data >::operator= (
            const SortableLinearContainer< Data > & )  [delete], [noexcept]
```

6.18.3.3 operator=() [2/2]

```
template<typename Data>
SortableLinearContainer & lasd::SortableLinearContainer< Data >::operator= (
            SortableLinearContainer< Data > && )  [delete], [noexcept]
```

6.18.3.4 operator==()

```
template<typename Data>
bool lasd::SortableLinearContainer< Data >::operator== (
            const SortableLinearContainer< Data > & con) const  [inline], [noexcept]
```

Operatore di confronto di uguaglianza.

Definition at line 125 of file linear.cpp.

6.18.3.5 partition()

```
template<typename Data>
ulong lasd::SortableLinearContainer< Data >::partition (
            ulong left,
            ulong right)  [protected], [noexcept]
```

Esegue la partizione per QuickSort.

Parameters

left	Indice di inizio.
right	Indice di fine.

Returns

Indice del pivot.

Definition at line 164 of file linear.cpp.

6.18.3.6 quickSort()

```
template<typename Data>
void lasd::SortableLinearContainer< Data >::quickSort (
            ulong left,
            ulong right)  [protected], [noexcept]
```

Esegue QuickSort ricorsivamente tra due indici.

Parameters

left	Indice di inizio.
right	Indice di fine.

Definition at line 145 of file linear.cpp.

6.18.3.7 Sort()

```
template<typename Data>
void lasd::SortableLinearContainer< Data >::Sort ()  [inline], [noexcept]
```

Ordina il contenitore secondo un algoritmo non specificato (QuickSort/Insertsort).

Definition at line 139 of file linear.cpp.

6.18.4 Member Data Documentation

6.18.4.1 size

```
template<typename Data>
ulong lasd::Container::size  [protected]
```

L'attributo size indica il numero di elementi presenti nel Container

Definition at line 15 of file container.hpp.

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

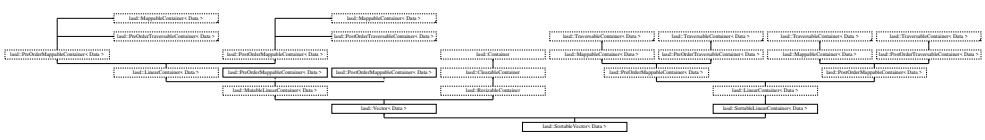
- Exercise1/container/linear.hpp
- Exercise1/container/linear.cpp

6.19 lasd::SortableVector< Data > Class Template Reference

Classe concreta che estende Vector con capacità di ordinamento.

```
#include <vector.hpp>
```

Inheritance diagram for lasd::SortableVector< Data >:



Public Member Functions

- SortableVector ()=default
- SortableVector (ulong dim)

  Costruttore con dimensione iniziale.
- SortableVector (const TraversableContainer< Data > &con)

  Costruttore da TraversableContainer.
- SortableVector (MappableContainer< Data > &&con)

  Costruttore da MappableContainer (move).
- SortableVector (const SortableVector< Data > &other)
- SortableVector (SortableVector< Data > &&other) noexcept
- virtual ~SortableVector ()=default
- SortableVector< Data > & operator= (const SortableVector< Data > &other)
- SortableVector< Data > & operator= (SortableVector< Data > &&other) noexcept

Public Member Functions inherited from lasd::Vector< Data >

- Vector ()=default

  Costruttore di default. Crea un vettore vuoto.
- Vector (ulong)

  Costruttore che inizializza il vettore con una data dimensione.
- Vector (const TraversableContainer< Data > &con)

  Costruttore che inizializza il vettore copiando da un TraversableContainer.
- Vector (MappableContainer< Data > &&con)

  Costruttore che inizializza il vettore muovendo da un MappableContainer.
- Vector (const Vector< Data > &)

  Costruttore di copia.
- Vector (Vector< Data > &&) noexcept

  Costruttore di spostamento.
- virtual ~Vector ()

  Distruttore.
- Vector< Data > & operator= (const Vector< Data > &)

  Assegnamento di copia.
- Vector< Data > & operator= (Vector< Data > &&) noexcept

  Assegnamento di spostamento.
- bool operator== (const Vector< Data > &) const noexcept

  Operatore di uguaglianza.
- bool operator!= (const Vector< Data > &) const noexcept

  Operatore di disuguaglianza.
- Data & operator[ ] (ulong) override

  Restituisce un riferimento modificabile all'elemento in posizione specifica.
- Data & Front () override

  Restituisce un riferimento modificabile al primo elemento del contenitore.
- Data & Back () override

  Restituisce un riferimento modificabile all'ultimo elemento del contenitore.
- const Data & operator[ ] (ulong) const override
- const Data & Front () const override

  Restituisce il primo elemento (costante).
- const Data & Back () const override

  Restituisce l'ultimo elemento (costante).
- void Resize (ulong) override

  Modifica la dimensione del vettore.
- void Clear () override

  Svuota il contenuto del vettore.

Public Member Functions inherited from lasd::MutableLinearContainer< Data >

- virtual ~MutableLinearContainer ()=default

  Distruttore virtuale di default.
- MutableLinearContainer & operator= (const MutableLinearContainer &)=delete
- MutableLinearContainer & operator= (MutableLinearContainer &&) noexcept=delete
- void Map (MapFun mapFun) override

  Applica una funzione a ciascun elemento (mappatura) in ordine indefinito.
- virtual void PreOrderMap (MapFun mapFun) override=0

  Applica una funzione a ciascun elemento (mappatura) in ordine PreOrder.
- virtual void PostOrderMap (MapFun mapFun) override=0

  Applica una funzione a ciascun elemento (mappatura) in ordine PostOrder.

Public Member Functions inherited from lasd::LinearContainer< Data >

- virtual ~LinearContainer ()=default

  Distruttore virtuale di default.
- LinearContainer & operator= (const LinearContainer &)=delete
- LinearContainer & operator= (LinearContainer &&) noexcept=delete
- bool operator== (const LinearContainer &) const noexcept

  Operatore di confronto di uguaglianza.
- bool operator!= (const LinearContainer &) const noexcept

  Operatore di confronto di disuguaglianza.
- virtual const Data & operator[ ] (const ulong int index) const =0

  Accesso in sola lettura all'elemento in posizione specifica.
- virtual Data & operator[ ] (const ulong int index)=0

  Accesso in lettura/scrittura all'elemento in posizione specifica.
- void Traverse (TraverseFun) const override

  Traversamento in ordine predefinito.
- void PreOrderTraverse (TraverseFun) const override

  Traversamento in ordine PreOrder.
- void PostOrderTraverse (TraverseFun) const override

  Traversamento in ordine PostOrder.

Public Member Functions inherited from lasd::PreOrderMappableContainer< Data >

- virtual ~PreOrderMappableContainer ()=default

  Distruttore virtuale di default.
- PreOrderMappableContainer & operator= (const PreOrderMappableContainer &)=delete
- PreOrderMappableContainer & operator= (PreOrderMappableContainer &&) noexcept=delete
- bool operator== (const PreOrderMappableContainer &) const noexcept=delete

  Operatore di confronto di uguaglianza disabilitato.
- bool operator!= (const PreOrderMappableContainer &) const noexcept=delete

  Operatore di confronto di disuguaglianza disabilitato.
- void Map (MapFun fun) override

  Applica la funzione mappante secondo la strategia PreOrder.

Public Member Functions inherited from lasd::MappableContainer< Data >

- virtual ~MappableContainer () noexcept=default

  Distruttore virtuale di default.
- MappableContainer & operator= (const MappableContainer &) noexcept=delete
- MappableContainer & operator= (MappableContainer &&) noexcept=delete
- bool operator== (const MappableContainer &) const noexcept=delete

  Operatore di confronto di uguaglianza disabilitato.
- bool operator!= (const MappableContainer &) const noexcept=delete

  Operatore di confronto di disuguaglianza disabilitato.

Public Member Functions inherited from lasd::TraversableContainer< Data >

- virtual ~TraversableContainer ()=default

  Distruttore virtuale di default.
- TraversableContainer & operator= (const TraversableContainer &)=delete
- TraversableContainer & operator= (TraversableContainer &&) noexcept=delete
- bool operator== (const TraversableContainer &) const noexcept=delete
- bool operator!= (const TraversableContainer &) const noexcept=delete
- template<typename Accumulator>
  Accumulator Fold (FoldFun< Accumulator > func, Accumulator base) const

  Esegue una riduzione (fold) sul contenitore.
- bool Exists (const Data &elem) const noexcept override

  Verifica se un elemento esiste nel contenitore.

Public Member Functions inherited from lasd::TestableContainer< Data >

- virtual ~TestableContainer ()=default

  Distruttore virtuale di default.
- TestableContainer & operator= (const TestableContainer &)=delete
- TestableContainer & operator= (TestableContainer &&) noexcept=delete
- bool operator== (const TestableContainer &) const noexcept=delete

  Operatore di confronto di uguaglianza disabilitato.
- bool operator!= (const TestableContainer &) const noexcept=delete

  Operatore di confronto di disuguaglianza disabilitato.

Public Member Functions inherited from lasd::Container

- virtual ~Container ()=default

  Destructor.
- Container & operator= (const Container &)=delete

  Copy assignment of abstract types is not possible.
- Container & operator= (Container &&) noexcept=delete

  Move assignment of abstract types is not possible.
- bool operator== (const Container &) const noexcept=delete
- bool operator!= (const Container &) const noexcept=delete
- virtual bool Empty () const noexcept

  La funzione Empty() controlla se la struttura è vuota (concrete function should not throw exceptions)
- virtual ulong Size () const noexcept

  La funzione Size() restituisce il numero di elementi presenti nella struttura (concrete function should not throw exceptions)

Public Member Functions inherited from lasd::PreOrderTraversableContainer< Data >

- virtual ~PreOrderTraversableContainer ()=default

  Distruttore virtuale di default.
- PreOrderTraversableContainer & operator= (const PreOrderTraversableContainer &)=delete
- PreOrderTraversableContainer & operator= (PreOrderTraversableContainer &&) noexcept=delete
- bool operator== (const PreOrderTraversableContainer &) const noexcept=delete
- bool operator!= (const PreOrderTraversableContainer &) const noexcept=delete
- template<typename Accumulator>
  Accumulator PreOrderFold (FoldFun< Accumulator > func, Accumulator base) const

  Esegue una riduzione in pre-ordine.
- void Traverse (TraverseFun func) const override

  Implementazione della traversata base come traversata in pre-ordine.

Public Member Functions inherited from lasd::PostOrderMappableContainer< Data >

- virtual ~PostOrderMappableContainer ()=default

  Distruttore virtuale di default.
- PostOrderMappableContainer & operator= (const PostOrderMappableContainer &)=delete
- PostOrderMappableContainer & operator= (PostOrderMappableContainer &&) noexcept=delete
- bool operator== (const PostOrderMappableContainer &) const noexcept=delete

  Operatore di confronto di uguaglianza disabilitato.
- bool operator!= (const PostOrderMappableContainer &) const noexcept=delete

  Operatore di confronto di disuguaglianza disabilitato.
- void Map (const MapFun fun) override

  Applica la funzione mappante secondo la strategia PostOrder.

Public Member Functions inherited from lasd::PostOrderTraversableContainer< Data >

- virtual ~PostOrderTraversableContainer ()=default

  Distruttore virtuale di default.
- PostOrderTraversableContainer & operator= (const PostOrderTraversableContainer &)=delete
- PostOrderTraversableContainer & operator= (PostOrderTraversableContainer &&) noexcept=delete
- bool operator== (const PostOrderTraversableContainer &) const noexcept=delete
- bool operator!= (const PostOrderTraversableContainer &) const noexcept=delete
- template<typename Accumulator>
  Accumulator PostOrderFold (FoldFun< Accumulator > func, Accumulator base) const

  Esegue una riduzione in post-ordine.
- void Traverse (TraverseFun func) const override

  Implementazione della traversata base come traversata in post-ordine.

Public Member Functions inherited from lasd::ResizableContainer

- virtual ~ResizableContainer ()=default

  Destructor.
- ResizableContainer & operator= (const ResizableContainer &Ccon) noexcept=delete

  Copy assignment.
- ResizableContainer & operator= (ResizableContainer &&Ccon) noexcept=delete

  Move assignment.
- bool operator== (const ResizableContainer &Ccon) const noexcept=delete
- bool operator!= (const ResizableContainer &Ccon) const noexcept=delete

Public Member Functions inherited from lasd::ClearableContainer

- virtual ~ClearableContainer ()=default

  Destructor.
- ClearableContainer & operator= (const ClearableContainer &)=delete

  Copy assignment.
- ClearableContainer & operator= (ClearableContainer &&) noexcept=delete

  Move assignment.
- bool operator== (const ClearableContainer &Ccon) const noexcept=delete

  Comparison operators.
- bool operator!= (const ClearableContainer &Ccon) const noexcept=delete

Public Member Functions inherited from lasd::SortableLinearContainer< Data >

- virtual ~SortableLinearContainer () noexcept=default

  Distruttore virtuale di default.
- SortableLinearContainer & operator= (const SortableLinearContainer &) noexcept=delete
- SortableLinearContainer & operator= (SortableLinearContainer &&) noexcept=delete
- bool operator== (const SortableLinearContainer &) const noexcept

  Operatore di confronto di uguaglianza.
- bool operator!= (const SortableLinearContainer &) const noexcept

  Operatore di confronto di disuguaglianza.
- void Sort () noexcept

  Ordina il contenitore secondo un algoritmo non specificato (QuickSort/Insertsort).

Additional Inherited Members

Public Types inherited from lasd::MappableContainer< Data >

- using MapFun = std::function<void(Data &)>

  Tipo di funzione mappante: accetta un riferimento modificabile a un elemento.

Public Types inherited from lasd::TraversableContainer< Data >

- using TraverseFun = std::function<void(const Data &)>

  Tipo funzione per la traversata (funzione unaria applicata a ogni dato).
- template<typename Accumulator>
  using FoldFun = std::function<Accumulator(const Data &, const Accumulator &)>

  Tipo funzione per fold (funzione binaria con accumulatore).

Public Types inherited from lasd::PreOrderTraversableContainer< Data >

- template<typename Accumulator>
  using FoldFun = typename TraversableContainer<Data>::FoldFun<Accumulator>

  Tipo funzione per fold in pre-ordine.

Public Types inherited from lasd::PostOrderTraversableContainer< Data >

- template<typename Accumulator>
  using FoldFun = typename TraversableContainer<Data>::FoldFun<Accumulator>

  Tipo funzione per fold in post-ordine.

Protected Member Functions inherited from lasd::TestableContainer< Data >

- TestableContainer ()=default

  Costruttore di default protetto.

Protected Member Functions inherited from lasd::Container

- Container ()=default

  Default constructor.

Protected Member Functions inherited from lasd::SortableLinearContainer< Data >

- void quickSort (ulong left, ulong right) noexcept

  Esegue QuickSort ricorsivamente tra due indici.
- ulong partition (ulong left, ulong right) noexcept

  Esegue la partizione per QuickSort.

Protected Attributes inherited from lasd::Vector< Data >

- Data * elements = nullptr

  Puntatore all'array degli elementi.
- ulong size

Protected Attributes inherited from lasd::LinearContainer< Data >

- ulong size

Protected Attributes inherited from lasd::Container

- ulong size = 0

Protected Attributes inherited from lasd::SortableLinearContainer< Data >

- ulong size

6.19.1 Detailed Description

template<typename Data>
class lasd::SortableVector< Data >

Classe concreta che estende Vector con capacità di ordinamento.

Template Parameters

Data	Tipo dei dati contenuti.

Definition at line 133 of file vector.hpp.

6.19.2 Constructor & Destructor Documentation

6.19.2.1 SortableVector() [1/6]

```
template<typename Data>
lasd::SortableVector< Data >::SortableVector ()  [default]
```

6.19.2.2 SortableVector() [2/6]

```
template<typename Data>
lasd::SortableVector< Data >::SortableVector (
            ulong dim)  [inline]
```

Costruttore con dimensione iniziale.

Parameters

dim	Dimensione iniziale del vettore.

Definition at line 146 of file vector.hpp.

6.19.2.3 SortableVector() [3/6]

```
template<typename Data>
lasd::SortableVector< Data >::SortableVector (
            const TraversableContainer< Data > & con)  [inline]
```

Costruttore da TraversableContainer.

Parameters

con	Contenitore traversabile da cui copiare.

Definition at line 152 of file vector.hpp.

6.19.2.4 SortableVector() [4/6]

```
template<typename Data>
lasd::SortableVector< Data >::SortableVector (
            MappableContainer< Data > && con)  [inline]
```

Costruttore da MappableContainer (move).

Parameters

con	Contenitore da muovere.

Definition at line 158 of file vector.hpp.

6.19.2.5 SortableVector() [5/6]

```
template<typename Data>
lasd::SortableVector< Data >::SortableVector (
            const SortableVector< Data > & other)  [inline]
```

Definition at line 161 of file vector.hpp.

6.19.2.6 SortableVector() [6/6]

```
template<typename Data>
lasd::SortableVector< Data >::SortableVector (
            SortableVector< Data > && other)  [inline], [noexcept]
```

Definition at line 164 of file vector.hpp.

6.19.2.7 ~SortableVector()

```
template<typename Data>
virtual lasd::SortableVector< Data >::~SortableVector ()  [virtual], [default]
```

6.19.3 Member Function Documentation

6.19.3.1 operator=() [1/2]

```
template<typename Data>
SortableVector< Data > & lasd::SortableVector< Data >::operator= (
            const SortableVector< Data > & other)  [inline]
```

Definition at line 170 of file vector.hpp.

6.19.3.2 operator=() [2/2]

```
template<typename Data>
SortableVector< Data > & lasd::SortableVector< Data >::operator= (
            SortableVector< Data > && other)  [inline], [noexcept]
```

Definition at line 177 of file vector.hpp.

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

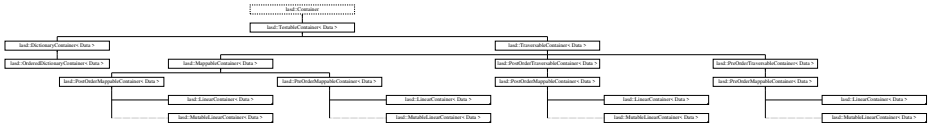
- Exercise1/vector/vector.hpp
- Exercise1/vector/vector.cpp

6.20 lasd::TestableContainer< Data > Class Template Reference

Classe astratta che estende Container con un metodo per il test di esistenza.

`#include <testable.hpp>`

Inheritance diagram for lasd::TestableContainer< Data >:



Public Member Functions

- virtual ~TestableContainer ()=default

  Distruttore virtuale di default.
- TestableContainer & operator= (const TestableContainer &)=delete
- TestableContainer & operator= (TestableContainer &&) noexcept=delete
- bool operator== (const TestableContainer &) const noexcept=delete

  Operatore di confronto di uguaglianza disabilitato.
- bool operator!= (const TestableContainer &) const noexcept=delete

  Operatore di confronto di disuguaglianza disabilitato.
- virtual bool Exists (const Data &dato) const noexcept=0

  Verifica se un elemento esiste nel contenitore.

Public Member Functions inherited from lasd::Container

- virtual ~Container ()=default

  Destructor.
- Container & operator= (const Container &)=delete

  Copy assignment of abstract types is not possible.
- Container & operator= (Container &&) noexcept=delete

  Move assignment of abstract types is not possible.
- bool operator== (const Container &) const noexcept=delete
- bool operator!= (const Container &) const noexcept=delete
- virtual bool Empty () const noexcept

  La funzione Empty() controlla se la struttura è vuota (concrete function should not throw exceptions)
- virtual ulong Size () const noexcept

  La funzione Size() restituisce il numero di elementi presenti nella struttura (concrete function should not throw exceptions)

Protected Member Functions

- TestableContainer ()=default

  Costruttore di default protetto.

Protected Member Functions inherited from lasd::Container

- Container ()=default

  Default constructor.

Additional Inherited Members

Protected Attributes inherited from lasd::Container

- ulong size = 0

6.20.1 Detailed Description

template<typename Data>
class lasd::TestableContainer< Data >

Classe astratta che estende Container con un metodo per il test di esistenza.

Template Parameters

Data	Tipo dei dati contenuti.

Definition at line 11 of file testable.hpp.

6.20.2 Constructor & Destructor Documentation

6.20.2.1 TestableContainer()

```
template<typename Data>
lasd::TestableContainer< Data >::TestableContainer ()  [protected], [default]
```

Costruttore di default protetto.

6.20.2.2 ~TestableContainer()

```
template<typename Data>
virtual lasd::TestableContainer< Data >::~TestableContainer ()  [virtual], [default]
```

Distruttore virtuale di default.

6.20.3 Member Function Documentation

6.20.3.1 Exists()

```
template<typename Data>
virtual bool lasd::TestableContainer< Data >::Exists (
            const Data & dato) const  [pure virtual], [noexcept]
```

Verifica se un elemento esiste nel contenitore.

Metodo puramente virtuale da implementare nelle classi concrete.

Parameters

in	*dato*	L'elemento da cercare.

Returns

true se l'elemento esiste nel contenitore, false altrimenti.

Implemented in lasd::TraversableContainer< Data >.

6.20.3.2 operator"!=()

```
template<typename Data>
bool lasd::TestableContainer< Data >::operator!= (
            const TestableContainer< Data > & ) const  [delete], [noexcept]
```

Operatore di confronto di disuguaglianza disabilitato.

6.20.3.3 operator=() [1/2]

```
template<typename Data>
TestableContainer & lasd::TestableContainer< Data >::operator= (
            const TestableContainer< Data > & )  [delete]
```

6.20.3.4 operator=() [2/2]

```
template<typename Data>
TestableContainer & lasd::TestableContainer< Data >::operator= (
            TestableContainer< Data > && )  [delete], [noexcept]
```

6.20.3.5 operator==()

```
template<typename Data>
bool lasd::TestableContainer< Data >::operator== (
            const TestableContainer< Data > & ) const  [delete], [noexcept]
```

Operatore di confronto di uguaglianza disabilitato.

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

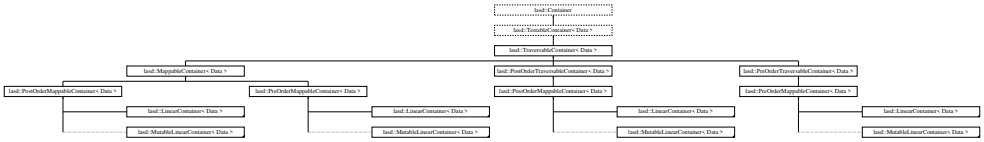
- Exercise1/container/testable.hpp

6.21 lasd::TraversableContainer< Data > Class Template Reference

Classe astratta di contenitore traversabile.

`#include <traversable.hpp>`

Inheritance diagram for lasd::TraversableContainer< Data >:



Public Types

- using TraverseFun = std::function<void(const Data &)>

  Tipo funzione per la traversata (funzione unaria applicata a ogni dato).
- template<typename Accumulator>
  using FoldFun = std::function<Accumulator(const Data &, const Accumulator &)>

  Tipo funzione per fold (funzione binaria con accumulatore).

Public Member Functions

- virtual ~TraversableContainer ()=default

  Distruttore virtuale di default.
- TraversableContainer & operator= (const TraversableContainer &)=delete
- TraversableContainer & operator= (TraversableContainer &&) noexcept=delete
- bool operator== (const TraversableContainer &) const noexcept=delete
- bool operator!= (const TraversableContainer &) const noexcept=delete
- virtual void Traverse (TraverseFun func) const =0

  Esegue una funzione su ogni elemento del contenitore.
- template<typename Accumulator>
  Accumulator Fold (FoldFun< Accumulator > func, Accumulator base) const

  Esegue una riduzione (fold) sul contenitore.
- bool Exists (const Data &elem) const noexcept override

  Verifica se un elemento esiste nel contenitore.

Public Member Functions inherited from lasd::TestableContainer< Data >

- virtual ~TestableContainer ()=default

  Distruttore virtuale di default.
- TestableContainer & operator= (const TestableContainer &)=delete
- TestableContainer & operator= (TestableContainer &&) noexcept=delete
- bool operator== (const TestableContainer &) const noexcept=delete

  Operatore di confronto di uguaglianza disabilitato.
- bool operator!= (const TestableContainer &) const noexcept=delete

  Operatore di confronto di disuguaglianza disabilitato.

Public Member Functions inherited from lasd::Container

- virtual ~Container ()=default

  Destructor.
- Container & operator= (const Container &)=delete

  Copy assignment of abstract types is not possible.
- Container & operator= (Container &&) noexcept=delete

  Move assignment of abstract types is not possible.
- bool operator== (const Container &) const noexcept=delete
- bool operator!= (const Container &) const noexcept=delete
- virtual bool Empty () const noexcept

  La funzione Empty() controlla se la struttura è vuota (concrete function should not throw exceptions)
- virtual ulong Size () const noexcept

  La funzione Size() restituisce il numero di elementi presenti nella struttura (concrete function should not throw exceptions)

Additional Inherited Members

Protected Member Functions inherited from lasd::TestableContainer< Data >

- TestableContainer ()=default

  Costruttore di default protetto.

Protected Member Functions inherited from lasd::Container

- Container ()=default

  Default constructor.

Protected Attributes inherited from lasd::Container

- ulong size = 0

6.21.1 Detailed Description

template<typename Data>
class lasd::TraversableContainer< Data >

Classe astratta di contenitore traversabile.

Template Parameters

Data	Tipo dei dati contenuti.

Definition at line 11 of file traversable.hpp.

6.21.2 Member Typedef Documentation

6.21.2.1 FoldFun

```
template<typename Data>
template<typename Accumulator>
using lasd::TraversableContainer< Data >::FoldFun = std::function<Accumulator(const Data &,
const Accumulator &)>
```

Tipo funzione per fold (funzione binaria con accumulatore).

Definition at line 37 of file traversable.hpp.

6.21.2.2 TraverseFun

```
template<typename Data>
using lasd::TraversableContainer< Data >::TraverseFun = std::function<void(const Data &)>
```

Tipo funzione per la traversata (funzione unaria applicata a ogni dato).

Definition at line 29 of file traversable.hpp.

6.21.3 Constructor & Destructor Documentation

6.21.3.1 ~TraversableContainer()

```
template<typename Data>
virtual lasd::TraversableContainer< Data >::~TraversableContainer ()  [virtual], [default]
```

Distruttore virtuale di default.

6.21.4 Member Function Documentation

6.21.4.1 Exists()

```
template<typename Data>
bool lasd::TraversableContainer< Data >::Exists (
            const Data & elem) const  [inline], [override], [virtual], [noexcept]
```

Verifica se un elemento esiste nel contenitore.

Parameters

elem	L'elemento da cercare.

Returns

true se l'elemento è presente, false altrimenti.

Implements lasd::TestableContainer< Data >.

Definition at line 16 of file traversable.cpp.

6.21.4.2 Fold()

```
template<typename Data>
template<typename Accumulator>
Accumulator lasd::TraversableContainer< Data, Accumulator >::Fold (
            FoldFun< Accumulator > func,
            Accumulator base) const  [inline]
```

Esegue una riduzione (fold) sul contenitore.

Template Parameters

Accumulator	Tipo dell'accumulatore.

Parameters

func	Funzione binaria che combina un elemento con l'accumulatore.
base	Valore iniziale dell'accumulatore.

Returns

Il risultato finale del fold.

Definition at line 4 of file traversable.cpp.

6.21.4.3 operator"!=()

```
template<typename Data>
bool lasd::TraversableContainer< Data >::operator!= (
            const TraversableContainer< Data > & ) const  [delete], [noexcept]
```

6.21.4.4 operator=() [1/2]

```
template<typename Data>
TraversableContainer & lasd::TraversableContainer< Data >::operator= (
            const TraversableContainer< Data > & )  [delete]
```

6.21.4.5 operator=() [2/2]

```
template<typename Data>
TraversableContainer & lasd::TraversableContainer< Data >::operator= (
            TraversableContainer< Data > && )  [delete], [noexcept]
```

6.21.4.6 operator==()

```
template<typename Data>
bool lasd::TraversableContainer< Data >::operator== (
            const TraversableContainer< Data > & ) const  [delete], [noexcept]
```

6.21.4.7 Traverse()

```
template<typename Data>
virtual void lasd::TraversableContainer< Data >::Traverse (
            TraverseFun func) const  [pure virtual]
```

Esegue una funzione su ogni elemento del contenitore.

Parameters

func	Funzione da applicare a ciascun elemento (passato per riferimento costante).

Implemented in lasd::LinearContainer< Data >, lasd::PostOrderTraversableContainer< Data >, and lasd::PreOrderTraversableConta

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

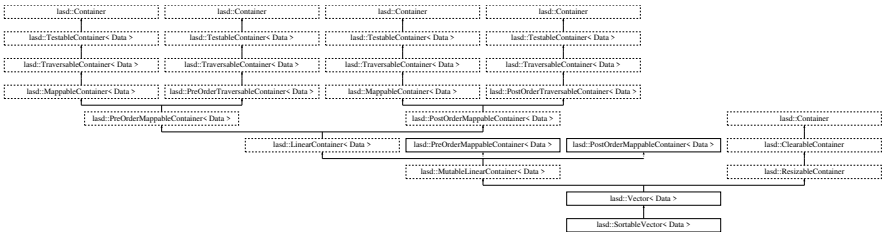
- Exercise1/container/traversable.hpp
- Exercise1/container/traversable.cpp

6.22 lasd::Vector< Data > Class Template Reference

Classe concreta che rappresenta un vettore dinamico di elementi.

`#include <vector.hpp>`

Inheritance diagram for lasd::Vector< Data >:



Public Member Functions

- Vector ()=default

  Costruttore di default. Crea un vettore vuoto.
- Vector (ulong)

  Costruttore che inizializza il vettore con una data dimensione.
- Vector (const TraversableContainer< Data > &con)

  Costruttore che inizializza il vettore copiando da un TraversableContainer.
- Vector (MappableContainer< Data > &&con)

  Costruttore che inizializza il vettore muovendo da un MappableContainer.
- Vector (const Vector< Data > &)

  Costruttore di copia.
- Vector (Vector< Data > &&) noexcept

  Costruttore di spostamento.
- virtual ~Vector ()

  Distruttore.
- Vector< Data > & operator= (const Vector< Data > &)

  Assegnamento di copia.
- Vector< Data > & operator= (Vector< Data > &&) noexcept

  Assegnamento di spostamento.
- bool operator== (const Vector< Data > &) const noexcept

Operatore di uguaglianza.

- bool operator!= (const Vector< Data > &) const noexcept

  Operatore di disuguaglianza.
- Data & operator[ ] (ulong) override

  Restituisce un riferimento modificabile all'elemento in posizione specifica.
- Data & Front () override

  Restituisce un riferimento modificabile al primo elemento del contenitore.
- Data & Back () override

  Restituisce un riferimento modificabile all'ultimo elemento del contenitore.
- const Data & operator[ ] (ulong) const override
- const Data & Front () const override

  Restituisce il primo elemento (costante).
- const Data & Back () const override

  Restituisce l'ultimo elemento (costante).
- void Resize (ulong) override

  Modifica la dimensione del vettore.
- void Clear () override

  Svuota il contenuto del vettore.

Public Member Functions inherited from lasd::MutableLinearContainer< Data >

- virtual ~MutableLinearContainer ()=default

  Distruttore virtuale di default.
- MutableLinearContainer & operator= (const MutableLinearContainer &)=delete
- MutableLinearContainer & operator= (MutableLinearContainer &&) noexcept=delete
- void Map (MapFun mapFun) override

  Applica una funzione a ciascun elemento (mappatura) in ordine indefinito.
- virtual void PreOrderMap (MapFun mapFun) override=0

  Applica una funzione a ciascun elemento (mappatura) in ordine PreOrder.
- virtual void PostOrderMap (MapFun mapFun) override=0

  Applica una funzione a ciascun elemento (mappatura) in ordine PostOrder.

Public Member Functions inherited from lasd::LinearContainer< Data >

- virtual ~LinearContainer ()=default

  Distruttore virtuale di default.
- LinearContainer & operator= (const LinearContainer &)=delete
- LinearContainer & operator= (LinearContainer &&) noexcept=delete
- bool operator== (const LinearContainer &) const noexcept

  Operatore di confronto di uguaglianza.
- bool operator!= (const LinearContainer &) const noexcept

  Operatore di confronto di disuguaglianza.
- virtual const Data & operator[ ] (const ulong int index) const =0

  Accesso in sola lettura all'elemento in posizione specifica.
- virtual Data & operator[ ] (const ulong int index)=0

  Accesso in lettura/scrittura all'elemento in posizione specifica.
- void Traverse (TraverseFun) const override

  Traversamento in ordine predefinito.
- void PreOrderTraverse (TraverseFun) const override

  Traversamento in ordine PreOrder.
- void PostOrderTraverse (TraverseFun) const override

  Traversamento in ordine PostOrder.

Public Member Functions inherited from lasd::PreOrderMappableContainer< Data >

- virtual ~PreOrderMappableContainer ()=default

  Distruttore virtuale di default.
- PreOrderMappableContainer & operator= (const PreOrderMappableContainer &)=delete
- PreOrderMappableContainer & operator= (PreOrderMappableContainer &&) noexcept=delete
- bool operator== (const PreOrderMappableContainer &) const noexcept=delete

  Operatore di confronto di uguaglianza disabilitato.
- bool operator!= (const PreOrderMappableContainer &) const noexcept=delete

  Operatore di confronto di disuguaglianza disabilitato.
- void Map (MapFun fun) override

  Applica la funzione mappante secondo la strategia PreOrder.

Public Member Functions inherited from lasd::MappableContainer< Data >

- virtual ~MappableContainer () noexcept=default

  Distruttore virtuale di default.
- MappableContainer & operator= (const MappableContainer &) noexcept=delete
- MappableContainer & operator= (MappableContainer &&) noexcept=delete
- bool operator== (const MappableContainer &) const noexcept=delete

  Operatore di confronto di uguaglianza disabilitato.
- bool operator!= (const MappableContainer &) const noexcept=delete

  Operatore di confronto di disuguaglianza disabilitato.

Public Member Functions inherited from lasd::TraversableContainer< Data >

- virtual ~TraversableContainer ()=default

  Distruttore virtuale di default.
- TraversableContainer & operator= (const TraversableContainer &)=delete
- TraversableContainer & operator= (TraversableContainer &&) noexcept=delete
- bool operator== (const TraversableContainer &) const noexcept=delete
- bool operator!= (const TraversableContainer &) const noexcept=delete
- template<typename Accumulator>
  Accumulator Fold (FoldFun< Accumulator > func, Accumulator base) const

  Esegue una riduzione (fold) sul contenitore.
- bool Exists (const Data &elem) const noexcept override

  Verifica se un elemento esiste nel contenitore.

Public Member Functions inherited from lasd::TestableContainer< Data >

- virtual ~TestableContainer ()=default

  Distruttore virtuale di default.
- TestableContainer & operator= (const TestableContainer &)=delete
- TestableContainer & operator= (TestableContainer &&) noexcept=delete
- bool operator== (const TestableContainer &) const noexcept=delete

  Operatore di confronto di uguaglianza disabilitato.
- bool operator!= (const TestableContainer &) const noexcept=delete

  Operatore di confronto di disuguaglianza disabilitato.

Public Member Functions inherited from lasd::Container

- virtual ~Container ()=default

  Destructor.
- Container & operator= (const Container &)=delete

  Copy assignment of abstract types is not possible.
- Container & operator= (Container &&) noexcept=delete

  Move assignment of abstract types is not possible.
- bool operator== (const Container &) const noexcept=delete
- bool operator!= (const Container &) const noexcept=delete
- virtual bool Empty () const noexcept

  La funzione Empty() controlla se la struttura è vuota (concrete function should not throw exceptions)
- virtual ulong Size () const noexcept

  La funzione Size() restituisce il numero di elementi presenti nella struttura (concrete function should not throw exceptions)

Public Member Functions inherited from lasd::PreOrderTraversableContainer< Data >

- virtual ~PreOrderTraversableContainer ()=default

  Distruttore virtuale di default.
- PreOrderTraversableContainer & operator= (const PreOrderTraversableContainer &)=delete
- PreOrderTraversableContainer & operator= (PreOrderTraversableContainer &&) noexcept=delete
- bool operator== (const PreOrderTraversableContainer &) const noexcept=delete
- bool operator!= (const PreOrderTraversableContainer &) const noexcept=delete
- template<typename Accumulator>
  Accumulator PreOrderFold (FoldFun< Accumulator > func, Accumulator base) const

  Esegue una riduzione in pre-ordine.
- void Traverse (TraverseFun func) const override

  Implementazione della traversata base come traversata in pre-ordine.

Public Member Functions inherited from lasd::PostOrderMappableContainer< Data >

- virtual ~PostOrderMappableContainer ()=default

  Distruttore virtuale di default.
- PostOrderMappableContainer & operator= (const PostOrderMappableContainer &)=delete
- PostOrderMappableContainer & operator= (PostOrderMappableContainer &&) noexcept=delete
- bool operator== (const PostOrderMappableContainer &) const noexcept=delete

  Operatore di confronto di uguaglianza disabilitato.
- bool operator!= (const PostOrderMappableContainer &) const noexcept=delete

  Operatore di confronto di disuguaglianza disabilitato.
- void Map (const MapFun fun) override

  Applica la funzione mappante secondo la strategia PostOrder.

Public Member Functions inherited from lasd::PostOrderTraversableContainer< Data >

- virtual ~PostOrderTraversableContainer ()=default

  Distruttore virtuale di default.
- PostOrderTraversableContainer & operator= (const PostOrderTraversableContainer &)=delete
- PostOrderTraversableContainer & operator= (PostOrderTraversableContainer &&) noexcept=delete
- bool operator== (const PostOrderTraversableContainer &) const noexcept=delete
- bool operator!= (const PostOrderTraversableContainer &) const noexcept=delete
- template<typename Accumulator>
  Accumulator PostOrderFold (FoldFun< Accumulator > func, Accumulator base) const

  Esegue una riduzione in post-ordine.
- void Traverse (TraverseFun func) const override

  Implementazione della traversata base come traversata in post-ordine.

Public Member Functions inherited from lasd::ResizableContainer

- virtual ~ResizableContainer ()=default

  Destructor.
- ResizableContainer & operator= (const ResizableContainer &Ccon) noexcept=delete

  Copy assignment.
- ResizableContainer & operator= (ResizableContainer &&Ccon) noexcept=delete

  Move assignment.
- bool operator== (const ResizableContainer &Ccon) const noexcept=delete
- bool operator!= (const ResizableContainer &Ccon) const noexcept=delete

Public Member Functions inherited from lasd::ClearableContainer

- virtual ~ClearableContainer ()=default

  Destructor.
- ClearableContainer & operator= (const ClearableContainer &)=delete

  Copy assignment.
- ClearableContainer & operator= (ClearableContainer &&) noexcept=delete

  Move assignment.
- bool operator== (const ClearableContainer &Ccon) const noexcept=delete

  Comparison operators.
- bool operator!= (const ClearableContainer &Ccon) const noexcept=delete

Protected Attributes

- Data * elements = nullptr

  Puntatore all'array degli elementi.
- ulong size

Protected Attributes inherited from lasd::LinearContainer< Data >

- ulong size

Protected Attributes inherited from lasd::Container

- ulong size = 0

Additional Inherited Members

Public Types inherited from lasd::MappableContainer< Data >

- using MapFun = std::function<void(Data &)>

  Tipo di funzione mappante: accetta un riferimento modificabile a un elemento.

Public Types inherited from lasd::TraversableContainer< Data >

- using TraverseFun = std::function<void(const Data &)>

  Tipo funzione per la traversata (funzione unaria applicata a ogni dato).
- template<typename Accumulator>
  using FoldFun = std::function<Accumulator(const Data &, const Accumulator &)>

  Tipo funzione per fold (funzione binaria con accumulatore).

Public Types inherited from lasd::PreOrderTraversableContainer< Data >

- template<typename Accumulator>
  using FoldFun = typename TraversableContainer<Data>::FoldFun<Accumulator>

  Tipo funzione per fold in pre-ordine.

Public Types inherited from lasd::PostOrderTraversableContainer< Data >

- template<typename Accumulator>
  using FoldFun = typename TraversableContainer<Data>::FoldFun<Accumulator>

  Tipo funzione per fold in post-ordine.

Protected Member Functions inherited from lasd::TestableContainer< Data >

- TestableContainer ()=default

  Costruttore di default protetto.

Protected Member Functions inherited from lasd::Container

- Container ()=default

  Default constructor.

6.22.1 Detailed Description

template<typename Data>
class lasd::Vector< Data >

Classe concreta che rappresenta un vettore dinamico di elementi.

Template Parameters

Data	Tipo dei dati contenuti.

Estende MutableLinearContainer e ResizableContainer.

Definition at line 16 of file vector.hpp.

6.22.2 Constructor & Destructor Documentation

6.22.2.1 Vector() [1/6]

```
template<typename Data>
lasd::Vector< Data >::Vector ()  [default]
```

Costruttore di default. Crea un vettore vuoto.

6.22.2.2 Vector() [2/6]

```
template<typename Data>
lasd::Vector< Data >::Vector (
            ulong dim)  [inline]
```

Costruttore che inizializza il vettore con una data dimensione.

Parameters

dim	Dimensione iniziale del vettore.

Definition at line 7 of file vector.cpp.

6.22.2.3 Vector() [3/6]

```
template<typename Data>
lasd::Vector< Data >::Vector (
            const TraversableContainer< Data > & con)  [inline]
```

Costruttore che inizializza il vettore copiando da un TraversableContainer.

Parameters

con	Contenitore traversabile da cui copiare gli elementi.

Definition at line 13 of file vector.cpp.

6.22.2.4 Vector() [4/6]

```
template<typename Data>
lasd::Vector< Data >::Vector (
            MappableContainer< Data > && con)  [inline]
```

Costruttore che inizializza il vettore muovendo da un MappableContainer.

Parameters

con	Contenitore mappabile da cui spostare gli elementi.

Definition at line 22 of file vector.cpp.

6.22.2.5 Vector() [5/6]

```
template<typename Data>
lasd::Vector< Data >::Vector (
            const Vector< Data > & vec)  [inline]
```

Costruttore di copia.

Definition at line 31 of file vector.cpp.

6.22.2.6 Vector() [6/6]

```
template<typename Data>
lasd::Vector< Data >::Vector (
            Vector< Data > && vec)  [inline], [noexcept]
```

Costruttore di spostamento.

Definition at line 37 of file vector.cpp.

6.22.2.7 ~Vector()

```
template<typename Data>
virtual lasd::Vector< Data >::~Vector ()  [virtual]
```

Distruttore.

6.22.3 Member Function Documentation

6.22.3.1 Back() [1/2]

```
template<typename Data>
const Data & lasd::Vector< Data >::Back () const  [inline], [override], [virtual]
```

Restituisce l'ultimo elemento (costante).

Returns

Riferimento costante all'ultimo elemento.

Reimplemented from lasd::LinearContainer< Data >.

Definition at line 158 of file vector.cpp.

6.22.3.2 Back() [2/2]

```
template<typename Data>
Data & lasd::Vector< Data >::Back ()  [inline], [override], [virtual]
```

Restituisce un riferimento modificabile all'ultimo elemento del contenitore.

Returns

Riferimento all'ultimo elemento.

Exceptions

std::length_error	se il contenitore è vuoto.

Implements lasd::MutableLinearContainer< Data >.

Definition at line 165 of file vector.cpp.

6.22.3.3 Clear()

```
template<typename Data>
void lasd::Vector< Data >::Clear ()  [inline], [override], [virtual]
```

Svuota il contenuto del vettore.

Reimplemented from lasd::ResizableContainer.

Definition at line 108 of file vector.cpp.

6.22.3.4 Front() [1/2]

```
template<typename Data>
const Data & lasd::Vector< Data >::Front () const  [inline], [override], [virtual]
```

Restituisce il primo elemento (costante).

Returns

Riferimento costante al primo elemento.

Reimplemented from lasd::LinearContainer< Data >.

Definition at line 143 of file vector.cpp.

6.22.3.5 Front() [2/2]

```
template<typename Data>
Data & lasd::Vector< Data >::Front ()  [inline], [override], [virtual]
```

Restituisce un riferimento modificabile al primo elemento del contenitore.

Returns

Riferimento al primo elemento.

Exceptions

std::length_error	se il contenitore è vuoto.

Implements lasd::MutableLinearContainer< Data >.

Definition at line 151 of file vector.cpp.

6.22.3.6 operator"!=()

```
template<typename Data>
bool lasd::Vector< Data >::operator!= (
            const Vector< Data > & vec) const  [inline], [noexcept]
```

Operatore di disuguaglianza.

Definition at line 80 of file vector.cpp.

6.22.3.7 operator=() [1/2]

```
template<typename Data>
Vector< Data > & lasd::Vector< Data >::operator= (
            const Vector< Data > & vec)  [inline]
```

Assegnamento di copia.

Definition at line 46 of file vector.cpp.

6.22.3.8 operator=() [2/2]

```
template<typename Data>
Vector< Data > & lasd::Vector< Data >::operator= (
            Vector< Data > && vec)  [inline], [noexcept]
```

Assegnamento di spostamento.

Definition at line 54 of file vector.cpp.

6.22.3.9 operator==()

```
template<typename Data>
bool lasd::Vector< Data >::operator== (
            const Vector< Data > & vec) const  [noexcept]
```

Operatore di uguaglianza.

Definition at line 62 of file vector.cpp.

6.22.3.10 operator[]() [1/2]

```
template<typename Data>
const Data & lasd::Vector< Data >::operator[] (
            ulong i) const  [inline], [override]
```

Definition at line 86 of file vector.cpp.

6.22.3.11 operator[]() [2/2]

```
template<typename Data>
Data & lasd::Vector< Data >::operator[] (
            ulong index)  [inline], [override], [virtual]
```

Restituisce un riferimento modificabile all'elemento in posizione specifica.

Parameters

index	Indice dell'elemento da accedere.

Returns

Riferimento all'elemento.

Exceptions

std::out_of_range	se l'indice è fuori dal range.

Implements lasd::MutableLinearContainer< Data >.

Definition at line 96 of file vector.cpp.

6.22.3.12 Resize()

```
template<typename Data>
void lasd::Vector< Data >::Resize (
            ulong s)  [override], [virtual]
```

Modifica la dimensione del vettore.

Parameters

newSize	Nuova dimensione desiderata.

Implements lasd::ResizableContainer.

Definition at line 116 of file vector.cpp.

6.22.4 Member Data Documentation

6.22.4.1 elements

```
template<typename Data>
Data* lasd::Vector< Data >::elements = nullptr  [protected]
```

Puntatore all'array degli elementi.

Definition at line 25 of file vector.hpp.

6.22.4.2 size

```
template<typename Data>
ulong lasd::Container::size  [protected]
```

L'attributo size indica il numero di elementi presenti nel Container

Definition at line 15 of file container.hpp.

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

- Exercise1/vector/vector.hpp
- Exercise1/vector/vector.cpp

Chapter 7

File Documentation

7.1 Exercise1/container/dictionary.cpp File Reference

Namespaces

- namespace lasd

7.2 dictionary.cpp

Go to the documentation of this file.

```
00001 namespace lasd
00002 {
00003     template <typename Data>
00004     inline bool DictionaryContainer<Data>::InsertAll(const TraversableContainer<Data> &con)
00005     {
00006         bool result = true;
00007         con.Traverse([this, &result](const Data &currData)
00008                      { result &= this->Insert(currData); });
00009         return result;
00010     }
00011
00012     template <typename Data>
00013     inline bool DictionaryContainer<Data>::InsertAll(MappableContainer<Data> &&con)
00014     {
00015         bool result = true;
00016         con.Map([this, &result](Data &currData)
00017                 { result &= this->Insert(std::move(currData)); });
00018         return result;
00019     }
00020
00021     template <typename Data>
00022     inline bool DictionaryContainer<Data>::RemoveAll(const TraversableContainer<Data> &con)
00023     {
00024         bool result = true;
00025         con.Traverse([this, &result](const Data &currData)
00026                      { result &= this->Remove(currData); });
00027         return result;
00028     }
00029
00030     template <typename Data>
00031     inline bool DictionaryContainer<Data>::InsertSome(const TraversableContainer<Data> &con)
00032     {
00033         bool result = false;
00034         con.Traverse([this, &result](const Data &currData)
00035                      { result |= this->Insert(currData); });
00036         return result;
00037     }
00038
00039     template <typename Data>
00040     inline bool DictionaryContainer<Data>::InsertSome(MappableContainer<Data> &&con)
00041     {
```

```
00042         bool result = false;
00043         con.Map([this, &result](Data &currData)
00044                 { result |= this->Insert(std::move(currData)); });
00045         return result;
00046     }
00047
00048     template <typename Data>
00049     inline bool DictionaryContainer<Data>::RemoveSome(const TraversableContainer<Data> &con)
00050     {
00051         bool result = false;
00052         con.Traverse([this, &result](const Data &currData)
00053                      { result |= this->Remove(currData); });
00054         return result;
00055     }
00056
00057 }
```

7.3 Exercise1/container/linear.cpp File Reference

Namespaces

- namespace lasd

7.4 linear.cpp

Go to the documentation of this file.

```
00001
00002 namespace lasd
00003 {
00004
00005     template <typename Data>
00006     inline bool LinearContainer<Data>::operator==(
00007         const LinearContainer<Data> &con) const noexcept
00008     {
00009         if (size != con.size)
00010         {
00011             return false;
00012         }
00013
00014         for (ulong i = 0; i < size; ++i)
00015         {
00016             if ((*this)[i] != con[i])
00017             {
00018                 return false;
00019             }
00020         }
00021         return true;
00022     }
00023
00024     template <typename Data>
00025     inline bool LinearContainer<Data>::operator!=(const LinearContainer<Data> &con) const noexcept
00026     {
00027         return !(*this == con);
00028     }
00029
00030     // Specific Methods
00031
00032     template <typename Data>
00033     inline const Data &LinearContainer<Data>::Front() const
00034     {
00035         if (size == 0)
00036         {
00037             throw std::length_error("Empty structure.");
00038         }
00039         return (*this)[0];
00040     }
00041
00042     template <typename Data>
00043     inline Data &LinearContainer<Data>::Front()
00044     {
00045         if (size != 0)
00046         {
00047             throw std::length_error("Empty structure.");
00048         }
```

```
00049         return (*this)[0];
00050     }
00051
00052     template <typename Data>
00053     inline const Data &LinearContainer<Data>::Back() const
00054     {
00055         if (size == 0)
00056         {
00057             throw std::length_error("Empty structure.");
00058         }
00059         return (*this)[size - 1];
00060     }
00061
00062     template <typename Data>
00063     inline Data &LinearContainer<Data>::Back()
00064     {
00065         if (size == 0)
00066         {
00067             throw std::length_error("Empty structure.");
00068         }
00069         return (*this)[size - 1];
00070     }
00071     // Overrided Methods
00072
00073     template <typename Data>
00074     inline void LinearContainer<Data>::Traverse(TraverseFun func) const
00075     {
00076         PreOrderTraverse(func);
00077     }
00078
00079     template <typename Data>
00080     inline void
00081     LinearContainer<Data>::PreOrderTraverse(const TraverseFun func) const
00082     {
00083         for (ulong i = 0; i < size; ++i)
00084         {
00085             func((*this)[i]);
00086         }
00087     }
00088
00089     template <typename Data>
00090     inline void
00091     LinearContainer<Data>::PostOrderTraverse(const TraverseFun func) const
00092     {
00093
00094         for (ulong i = size; i > 0;)
00095         {
00096             func((*this)[--i]);
00097         }
00098     }
00099
00100     template <typename Data>
00101     inline void LinearContainer<Data>::Map(MapFun func)
00102     {
00103         PreOrderMap(func);
00104     }
00105
00106     template <typename Data>
00107     inline void LinearContainer<Data>::PreOrderMap(MapFun func)
00108     {
00109         for (ulong i = 0; i < size; ++i)
00110         {
00111             func((*this)[i]);
00112         }
00113     }
00114
00115     template <typename Data>
00116     inline void LinearContainer<Data>::PostOrderMap(MapFun func)
00117     {
00118         for (ulong i = size; i > 0;)
00119         {
00120             func((*this)[--i]);
00121         }
00122     }
00123
00124     template <typename Data>
00125     inline bool SortableLinearContainer<Data>::operator==(const SortableLinearContainer<Data> &con)
    const noexcept
00126     {
00127         return LinearContainer<Data>::operator==(con);
00128     }
00129
00130     template <typename Data>
00131     inline bool SortableLinearContainer<Data>::operator!=(const SortableLinearContainer<Data> &con)
    const noexcept
00132     {
00133         return !(*this == con);
```

```
00134     }
00135
00136     // Specific Methods
00137
00138     template <typename Data>
00139     inline void SortableLinearContainer<Data>::Sort() noexcept
00140     {
00141         quickSort(0, size - 1);
00142     }
00143
00144     template <typename Data>
00145     void SortableLinearContainer<Data>::quickSort(ulong p, ulong r) noexcept
00146     {
00147         if (p >= r)
00148         {
00149             return;
00150         }
00151
00152         if (r - p < 16)
00153         {
00154             insertionSort(p, r);
00155             return;
00156         }
00157
00158         ulong h = partition(p, r);
00159         quickSort(p, h);
00160         quickSort(h + 1, r);
00161     }
00162
00163     template <typename Data>
00164     ulong SortableLinearContainer<Data>::partition(ulong p, ulong r) noexcept
00165     {
00166         ulong i = p - 1;
00167         ulong j = r + 1;
00168
00169         std::swap((*this)[p], (*this)[randomMedian(p, r)]);
00170         Data pivot = (*this)[p];
00171
00172         do
00173         {
00174             do
00175             {
00176                 --j;
00177             } while (pivot < (*this)[j]);
00178
00179             do
00180             {
00181                 ++i;
00182             } while (pivot > (*this)[i]);
00183
00184             if (i < j)
00185             {
00186                 std::swap((*this)[i], (*this)[j]);
00187             }
00188         } while (i < j);
00189
00190         return j;
00191     }
00192 }
```

7.5 Exercise1/container/mappable.cpp File Reference

Namespaces

- namespace lasd

7.6 mappable.cpp

Go to the documentation of this file.

```
00001
00002 namespace lasd
00003 {
00004
00005     template <typename Data>
```

```
00006     inline void PreOrderMappableContainer<Data>::Map(MapFun fun)
00007     {
00008         PreOrderMap(fun);
00009     }
00010
00011     template <typename Data>
00012     inline void PostOrderMappableContainer<Data>::Map(MapFun fun)
00013     {
00014         PostOrderMap(fun);
00015     }
00016
00017 }
```

7.7 Exercise1/container/traversable.cpp File Reference

Namespaces

- namespace lasd

7.8 traversable.cpp

Go to the documentation of this file.

```
00001 namespace lasd
00002 {
00003     template <typename Data, typename Accumulator>
00004     inline Accumulator TraversableContainer<Data>::Fold(FoldFun<Accumulator> func, Accumulator base)
      const
00005     {
00006         Traverse(
00007             [&base, func](const Data &currData)
00008             {
00009                 base = func(currData, base);
00010             }
00011         );
00012         return base;
00013     };
00014
00015     template <typename Data>
00016     inline bool TraversableContainer<Data>::Exists(const Data &data) const noexcept
00017     {
00018         bool exists = false;
00019         Traverse(
00020             [data, &exists](const Data &currData)
00021             {
00022                 exists |= (data == currData);
00023             }
00024         );
00025         return exists;
00026     }
00027
00028     template <typename Data>
00029     inline void PreOrderTraversableContainer<Data>::Traverse(TraverseFun func) const
00030     {
00031         PreOrderTraverse(func);
00032     }
00033
00034     template <typename Data>
00035     template <typename Accumulator>
00036     inline Accumulator PreOrderTraversableContainer<Data>::PreOrderFold(FoldFun<Accumulator> func,
      Accumulator base) const
00037     {
00038         PreOrderTraverse(
00039             [&base, func](const Data &currData)
00040             {
00041                 base = func(currData, base);
00042             }
00043         );
00044         return base;
00045     };
00046
00047     template <typename Data>
00048     inline void PostOrderTraversableContainer<Data>::Traverse(TraverseFun func) const
00049     {
00050         PostOrderTraverse(func);
```

```
00051     }
00052
00053     template <typename Data>
00054     template <typename Accumulator>
00055     inline Accumulator PostOrderTraversableContainer<Data>::PostOrderFold(FoldFun<Accumulator> func,
      Accumulator base) const
00056     {
00057         PostOrderTraverse(
00058             [&base, &func](const Data &currData)
00059             {
00060                 base = func(currData, base);
00061             }
00062         );
00063         return base;
00064     };
00065
00066 }
```

7.9 Exercise1/list/list.cpp File Reference

Namespaces

- namespace lasd

7.10 list.cpp

Go to the documentation of this file.

```
00001
00002 namespace lasd {
00003
00004 /* ************************************************************************** */
00005
00006 // ...
00007
00008 /* ************************************************************************** */
00009
00010 }
```

7.11 Exercise1/main.cpp File Reference

```
#include "zlasdtest/test.hpp"
#include "zmytest/test.hpp"
#include <iostream>
```

Functions

- int main ()

7.11.1 Function Documentation

7.11.1.1 main()

```
int main ()
```

Definition at line 12 of file main.cpp.

7.12 main.cpp

Go to the documentation of this file.

```
00001
00002 #include "zlasdtest/test.hpp"
00003
00004 #include "zmytest/test.hpp"
00005
00006 /* ************************************************************************** */
00007
00008 #include <iostream>
00009
00010 /* ************************************************************************** */
00011
00012 int main() {
00013   std::cout << "LASD Libraries 2025" << std::endl;
00014   lasdtest();
00015   return 0;
00016 }
```

7.13 Exercise1/set/lst/setlst.cpp File Reference

Namespaces

- namespace lasd

7.14 setlst.cpp

Go to the documentation of this file.

```
00001
00002 namespace lasd {
00003
00004 /* ************************************************************************** */
00005
00006 // ...
00007
00008 /* ************************************************************************** */
00009
00010 }
```

7.15 Exercise1/set/lst/setlst.hpp File Reference

```
#include "../set.hpp"
#include "../../list/list.hpp"
#include "setlst.cpp"
```

Classes

- class lasd::SetLst< Data >

Namespaces

- namespace lasd

7.16 setlst.hpp

Go to the documentation of this file.

```
00001
00002 #ifndef SETLST_HPP
00003 #define SETLST_HPP
00004
00005 /* ************************************************************************** */
00006
00007 #include "../set.hpp"
00008 #include "../../list/list.hpp"
00009
00010 /* ************************************************************************** */
00011
00012 namespace lasd {
00013
00014 /* ************************************************************************** */
00015
00016 template <typename Data>
00017 class SetLst {
00018   // Must extend Set<Data>,
00019   //             List<Data>
00020
00021 private:
00022
00023   // ...
00024
00025 protected:
00026
00027   // using Container::???;
00028
00029   // ...
00030
00031 public:
00032
00033   // Default constructor
00034   // SetLst() specifiers;
00035
00036   /* ************************************************************************ */
00037
00038   // Specific constructors
00039   // SetLst(argument) specifiers; // A set obtained from a TraversableContainer
00040   // SetLst(argument) specifiers; // A set obtained from a MappableContainer
00041
00042   /* ************************************************************************ */
00043
00044   // Copy constructor
00045   // SetLst(argument) specifiers;
00046
00047   // Move constructor
00048   // SetLst(argument) specifiers;
00049
00050   /* ************************************************************************ */
00051
00052   // Destructor
00053   // ~SetLst() specifiers;
00054
00055   /* ************************************************************************ */
00056
00057   // Copy assignment
00058   // type operator=(argument) specifiers;
00059
00060   // Move assignment
00061   // type operator=(argument) specifiers;
00062
00063   /* ************************************************************************ */
00064
00065   // Comparison operators
00066   // type operator==(argument) specifiers;
00067   // type operator!=(argument) specifiers;
00068
00069   /* ************************************************************************ */
00070
00071   // Specific member functions (inherited from OrderedDictionaryContainer)
00072
00073   // type Min(argument) specifiers; // Override OrderedDictionaryContainer member (concrete function
      must throw std::length_error when empty)
00074   // type MinNRemove(argument) specifiers; // Override OrderedDictionaryContainer member (concrete
      function must throw std::length_error when empty)
00075   // type RemoveMin(argument) specifiers; // Override OrderedDictionaryContainer member (concrete
      function must throw std::length_error when empty)
00076
00077   // type Max(argument) specifiers; // Override OrderedDictionaryContainer member (concrete function
      must throw std::length_error when empty)
```

```
00078   // type MaxNRemove(argument) specifiers; // Override OrderedDictionaryContainer member (concrete
      function must throw std::length_error when empty)
00079   // type RemoveMax(argument) specifiers; // Override OrderedDictionaryContainer member (concrete
      function must throw std::length_error when empty)
00080
00081   // type Predecessor(argument) specifiers; // Override OrderedDictionaryContainer member (concrete
      function must throw std::length_error when not found)
00082   // type PredecessorNRemove(argument) specifiers; // Override OrderedDictionaryContainer member
      (concrete function must throw std::length_error when not found)
00083   // type RemovePredecessor(argument) specifiers; // Override OrderedDictionaryContainer member
      (concrete function must throw std::length_error when not found)
00084
00085   // type Successor(argument) specifiers; // Override OrderedDictionaryContainer member (concrete
      function must throw std::length_error when not found)
00086   // type SuccessorNRemove(argument) specifiers; // Override OrderedDictionaryContainer member
      (concrete function must throw std::length_error when not found)
00087   // type RemoveSuccessor(argument) specifiers; // Override OrderedDictionaryContainer member
      (concrete function must throw std::length_error when not found)
00088
00089   /* ************************************************************************ */
00090
00091   // Specific member functions (inherited from DictionaryContainer)
00092
00093   // type Insert(argument) specifiers; // Override DictionaryContainer member (copy of the value)
00094   // type Insert(argument) specifiers; // Override DictionaryContainer member (move of the value)
00095   // type Remove(argument) specifiers; // Override DictionaryContainer member
00096
00097   /* ************************************************************************ */
00098
00099   // Specific member functions (inherited from LinearContainer)
00100
00101   // type operator[](argument) specifiers; // Override LinearContainer member (must throw
      std::out_of_range when out of range)
00102
00103   /* ************************************************************************ */
00104
00105   // Specific member function (inherited from TestableContainer)
00106
00107   // type Exists(argument) specifiers; // Override TestableContainer member
00108
00109   /* ************************************************************************ */
00110
00111   // Specific member function (inherited from ClearableContainer)
00112
00113   // type Clear() specifiers; // Override ClearableContainer member
00114
00115 protected:
00116
00117   // Auxiliary functions, if necessary!
00118
00119 };
00120
00121 /* ************************************************************************ */
00122
00123 }
00124
00125 #include "setlst.cpp"
00126
00127 #endif
```

7.17 Exercise1/set/vec/setvec.cpp File Reference

Namespaces

- namespace lasd

7.18 setvec.cpp

Go to the documentation of this file.

```
00001
00002 namespace lasd {
00003
00004 /* ************************************************************************ */
00005
```

```
00006 // ...
00007
00008 /* ************************************************************************ */
00009
00010 }
```

7.19 Exercise1/set/vec/setvec.hpp File Reference

```
#include "../set.hpp"
#include "../../vector/vector.hpp"
#include "setvec.cpp"
```

Classes

- class lasd::SetVec< Data >

Namespaces

- namespace lasd

7.20 setvec.hpp

Go to the documentation of this file.

```
00001
00002 #ifndef SETVEC_HPP
00003 #define SETVEC_HPP
00004
00005 /* ************************************************************************ */
00006
00007 #include "../set.hpp"
00008 #include "../../vector/vector.hpp"
00009
00010 /* ************************************************************************ */
00011
00012 namespace lasd {
00013
00014 /* ************************************************************************ */
00015
00016 template <typename Data>
00017 class SetVec {
00018   // Must extend Set<Data>,
00019   //             ResizableContainer
00020
00021 private:
00022
00023   // ...
00024
00025 protected:
00026
00027   // using Container::???;
00028
00029   // ...
00030
00031 public:
00032
00033   // Default constructor
00034   // SetVec() specifiers;
00035
00036   /* ************************************************************************ */
00037
00038   // Specific constructors
00039   // SetVec(argument) specifiers; // A set obtained from a TraversableContainer
00040   // SetVec(argument) specifiers; // A set obtained from a MappableContainer
00041
00042   /* ************************************************************************ */
```

```
00043
00044   // Copy constructor
00045   // SetVec(argument) specifiers;
00046
00047   // Move constructor
00048   // SetVec(argument) specifiers;
00049
00050   /* ************************************************************************ */
00051
00052   // Destructor
00053   // ~SetVec() specifiers;
00054
00055   /* ************************************************************************ */
00056
00057   // Copy assignment
00058   // type operator=(argument) specifiers;
00059
00060   // Move assignment
00061   // type operator=(argument) specifiers;
00062
00063   /* ************************************************************************ */
00064
00065   // Comparison operators
00066   // type operator==(argument) specifiers;
00067   // type operator!=(argument) specifiers;
00068
00069   /* ************************************************************************ */
00070
00071   // Specific member functions (inherited from OrderedDictionaryContainer)
00072
00073   // type Min(argument) specifiers; // Override OrderedDictionaryContainer member (concrete function
      must throw std::length_error when empty)
00074   // type MinNRemove(argument) specifiers; // Override OrderedDictionaryContainer member (concrete
      function must throw std::length_error when empty)
00075   // type RemoveMin(argument) specifiers; // Override OrderedDictionaryContainer member (concrete
      function must throw std::length_error when empty)
00076
00077   // type Max(argument) specifiers; // Override OrderedDictionaryContainer member (concrete function
      must throw std::length_error when empty)
00078   // type MaxNRemove(argument) specifiers; // Override OrderedDictionaryContainer member (concrete
      function must throw std::length_error when empty)
00079   // type RemoveMax(argument) specifiers; // Override OrderedDictionaryContainer member (concrete
      function must throw std::length_error when empty)
00080
00081   // type Predecessor(argument) specifiers; // Override OrderedDictionaryContainer member (concrete
      function must throw std::length_error when not found)
00082   // type PredecessorNRemove(argument) specifiers; // Override OrderedDictionaryContainer member
      (concrete function must throw std::length_error when not found)
00083   // type RemovePredecessor(argument) specifiers; // Override OrderedDictionaryContainer member
      (concrete function must throw std::length_error when not found)
00084
00085   // type Successor(argument) specifiers; // Override OrderedDictionaryContainer member (concrete
      function must throw std::length_error when not found)
00086   // type SuccessorNRemove(argument) specifiers; // Override OrderedDictionaryContainer member
      (concrete function must throw std::length_error when not found)
00087   // type RemoveSuccessor(argument) specifiers; // Override OrderedDictionaryContainer member
      (concrete function must throw std::length_error when not found)
00088
00089   /* ************************************************************************ */
00090
00091   // Specific member functions (inherited from DictionaryContainer)
00092
00093   // type Insert(argument) specifiers; // Override DictionaryContainer member (copy of the value)
00094   // type Insert(argument) specifiers; // Override DictionaryContainer member (move of the value)
00095   // type Remove(argument) specifiers; // Override DictionaryContainer member
00096
00097   /* ************************************************************************ */
00098
00099   // Specific member functions (inherited from LinearContainer)
00100
00101   // type operator[](argument) specifiers; // Override LinearContainer member (must throw
      std::out_of_range when out of range)
00102
00103   /* ************************************************************************ */
00104
00105   // Specific member function (inherited from TestableContainer)
00106
00107   // type Exists(argument) specifiers; // Override TestableContainer member
00108
00109   /* ************************************************************************ */
00110
00111   // Specific member function (inherited from ClearableContainer)
00112
00113   // type Clear() specifiers; // Override ClearableContainer member
00114
00115 protected:
00116
```

```
00117   // Auxiliary functions, if necessary!
00118
00119 };
00120
00121 /* ********************************************************************** */
00122
00123 }
00124
00125 #include "setvec.cpp"
00126
00127 #endif
```

7.21 Exercise1/vector/vector.cpp File Reference

```
#include <algorithm>
```

Namespaces

- namespace lasd

7.22 vector.cpp

Go to the documentation of this file.

```
00001
00002 #include <algorithm>
00003 namespace lasd
00004 {
00005
00006     template <typename Data>
00007     inline Vector<Data>::Vector(ulong dim) {
00008         this->size = dim;
00009         this->elements = new Data[dim]();
00010     }
00011
00012     template <typename Data>
00013     inline Vector<Data>::Vector(const TraversableContainer<Data> &container) :
      Vector(container.Size())
00014     {
00015         ulong i{0};
00016         container.Traverse(
00017             [this, &i](const Data &currData)
00018             { elements[i++] = currData; });
00019     }
00020
00021     template <typename Data>
00022     inline Vector<Data>::Vector(MappableContainer<Data> &&container) : Vector(container.Size())
00023     {
00024         ulong i{0};
00025         container.Map(
00026             [this, &i](Data &currData)
00027             { std::swap(elements[i++], currData); });
00028     }
00029
00030     template <typename Data>
00031     inline Vector<Data>::Vector(const Vector<Data> &vec) : Vector(vec.size)
00032     {
00033         std::uninitialized_copy(vec.elements, vec.elements + size, elements);
00034     }
00035
00036     template <typename Data>
00037     inline Vector<Data>::Vector(Vector<Data> &&vec) noexcept
00038     {
00039         std::swap(size, vec.size);
00040         std::swap(elements, vec.elements);
00041     }
00042
00043     // Operators
00044
00045     template <typename Data>
```

```
00046     inline Vector<Data> &Vector<Data>::operator=(const Vector<Data> &vec)
00047     {
00048         Vector<Data> temp{vec};
00049         std::swap(temp, *this);
00050         return *this;
00051     }
00052
00053     template <typename Data>
00054     inline Vector<Data> &Vector<Data>::operator=(Vector<Data> &&vec) noexcept
00055     {
00056         std::swap(size, vec.size);
00057         std::swap(elements, vec.elements);
00058         return *this;
00059     }
00060
00061     template <typename Data>
00062     bool Vector<Data>::operator==(const Vector<Data> &vec) const noexcept
00063     {
00064         if (size != vec.size)
00065         {
00066             return false;
00067         }
00068
00069         for (ulong i = 0; i < size; i++)
00070         {
00071             if (elements[i] != vec[i])
00072             {
00073                 return false;
00074             }
00075         }
00076         return true;
00077     }
00078
00079     template <typename Data>
00080     inline bool Vector<Data>::operator!=(const Vector<Data> &vec) const noexcept
00081     {
00082         return !(*this == vec);
00083     }
00084
00085     template <typename Data>
00086     inline const Data &Vector<Data>::operator[](ulong i) const
00087     {
00088         if (i >= size)
00089         {
00090             throw std::out_of_range("This Vector has not that many elements");
00091         }
00092         return elements[i];
00093     }
00094
00095     template <typename Data>
00096     inline Data &Vector<Data>::operator[](ulong i)
00097     {
00098         if (i >= size)
00099         {
00100             throw std::out_of_range("This Vector has not that many elements");
00101         }
00102         return elements[i];
00103     }
00104
00105     // Overrided Methods
00106
00107     template <typename Data>
00108     inline void Vector<Data>::Clear()
00109     {
00110         delete[] elements;
00111         size = 0;
00112         elements = nullptr;
00113     }
00114
00115     template <typename Data>
00116     void Vector<Data>::Resize(ulong s)
00117     {
00118         if (s == size)
00119         {
00120             return;
00121         }
00122
00123         if (s == 0)
00124         {
00125             Clear();
00126             return;
00127         }
00128
00129         Data *temp{new Data[s]{}};
00130
00131         ulong min{std::min(s, size)};
00132
```

```
00133         for (ulong i{0}; i < min; ++i)
00134             std::swap(elements[i], temp[i]);
00135 
00136         std::swap(elements, temp);
00137         delete[] temp;
00138 
00139         size = s;
00140     }
00141 
00142     template <typename Data>
00143     inline const Data &Vector<Data>::Front() const
00144     {
00145         if (size != 0)
00146             return elements[0];
00147         throw std::length_error("The Vector is empty");
00148     }
00149 
00150     template <typename Data>
00151     inline Data &Vector<Data>::Front()
00152     {
00153         if (size != 0)
00154             return elements[0];
00155         throw std::length_error("The Vector is empty");
00156     }
00157     template <typename Data>
00158     inline const Data &Vector<Data>::Back() const
00159     {
00160         if (size != 0)
00161             return elements[size - 1];
00162         throw std::length_error("The Vector is empty");
00163     }
00164     template <typename Data>
00165     inline Data &Vector<Data>::Back()
00166     {
00167         if (size != 0)
00168             return elements[size - 1];
00169         throw std::length_error("The Vector is empty");
00170     }
00171 
00172     /* ********************************************************************** */
00173     template <typename Data>
00174     inline SortableVector<Data> &
00175     SortableVector<Data>::operator=(const SortableVector<Data> &vec)
00176     {
00177         Vector<Data>::operator=(vec);
00178         return *this;
00179     }
00180 
00181     template <typename Data>
00182     inline SortableVector<Data> &
00183     SortableVector<Data>::operator=(SortableVector<Data> &&vec) noexcept
00184     {
00185         Vector<Data>::operator=(std::move(vec));
00186         return *this;
00187     }
00188 
00189 }
```

7.23 Exercise1/zlasdtest/container/container.cpp File Reference

```
#include <iostream>
#include "../../container/container.hpp"
```

Functions

- void Empty (uint &testnum, uint &testerr, const lasd::Container &con, bool chk)
- void Size (uint &testnum, uint &testerr, const lasd::Container &con, bool chk, ulong siz)
- int FoldParity (const int &dat, const int &acc)
- std::string FoldStringConcatenate (const std::string &dat, const std::string &acc)
- void MapStringAppend (std::string &dat, const std::string &par)
- void MapStringNonEmptyAppend (std::string &dat, const std::string &par)

7.23.1 Function Documentation

7.23.1.1 Empty()

```
void Empty (
            uint & testnum,
            uint & testerr,
            const lasd::Container & con,
            bool chk)
```

Definition at line 12 of file container.cpp.

7.23.1.2 FoldParity()

```
int FoldParity (
            const int & dat,
            const int & acc)
```

Definition at line 32 of file container.cpp.

7.23.1.3 FoldStringConcatenate()

```
std::string FoldStringConcatenate (
            const std::string & dat,
            const std::string & acc)
```

Definition at line 36 of file container.cpp.

7.23.1.4 MapStringAppend()

```
void MapStringAppend (
            std::string & dat,
            const std::string & par)
```

Definition at line 46 of file container.cpp.

7.23.1.5 MapStringNonEmptyAppend()

```
void MapStringNonEmptyAppend (
            std::string & dat,
            const std::string & par)
```

Definition at line 50 of file container.cpp.

7.23.1.6 Size()

```
void Size (
            uint & testnum,
            uint & testerr,
            const lasd::Container & con,
            bool chk,
            ulong siz)
```

Definition at line 20 of file container.cpp.

7.24 container.cpp

Go to the documentation of this file.

```
00001
00002 #include <iostream>
00003
00004 /* ********************************************************************** */
00005
00006 #include "../../container/container.hpp"
00007
00008 /* ********************************************************************** */
00009
00010 // Container member functions!
00011
00012 void Empty(uint & testnum, uint & testerr, const lasd::Container & con, bool chk) {
00013   bool tst;
00014   testnum++;
00015   std::cout « " " « testnum « " (" « testerr « ") The container is " « ((tst = con.Empty()) ? "" :
      "not ") « "empty: ";
00016   std::cout « ((tst = (tst == chk)) ? "Correct" : "Error") « "!" « std::endl;
00017   testerr += (1 - (uint) tst);
00018 }
00019
00020 void Size(uint & testnum, uint & testerr, const lasd::Container & con, bool chk, ulong siz) {
00021   bool tst;
00022   testnum++;
00023   std::cout « " " « testnum « " (" « testerr « ") The container has size " « con.Size() « ": ";
00024   std::cout « ((tst = ((con.Size() == siz) == chk)) ? "Correct" : "Error") « "!" « std::endl;
00025   testerr += (1 - (uint) tst);
00026 }
00027
00028 /* ********************************************************************** */
00029
00030 // Auxiliary functions for TraversableContainer!
00031
00032 int FoldParity(const int & dat, const int & acc) {
00033   return ((acc + dat) % 2);
00034 }
00035
00036 std::string FoldStringConcatenate(const std::string & dat, const std::string & acc) {
00037   std::string newstr = acc;
00038   newstr.append(dat);
00039   return newstr;
00040 }
00041
00042 /* ********************************************************************** */
00043
00044 // Auxiliary functions for MappableContainer!
00045
00046 void MapStringAppend(std::string & dat, const std::string & par) {
00047   dat.append(par);
00048 }
00049
00050 void MapStringNonEmptyAppend(std::string & dat, const std::string & par) {
00051   if (!dat.empty()) { dat.append(par); }
00052 }
00053
00054 /* ********************************************************************** */
```

7.25 Exercise1/container/container.hpp File Reference

Classes

- class lasd::Container
- class lasd::ClearableContainer
- class lasd::ResizableContainer

Namespaces

- namespace lasd

7.26 container.hpp

Go to the documentation of this file.

```
00001
00002 #ifndef CONTAINER_HPP
00003 #define CONTAINER_HPP
00004
00005 namespace lasd
00006 {
00007
00011   class Container
00012   {
00013
00014   protected:
00015     ulong size = 0;
00016
00018     Container() = default;
00019
00020   public:
00022     virtual ~Container() = default;
00023
00025     Container &operator=(const Container &) = delete;
00027     Container &operator=(Container &&) noexcept = delete;
00028
00029     bool operator==(const Container &) const noexcept = delete;
00030     bool operator!=(const Container &) const noexcept = delete;
00031
00033
00036     inline virtual bool Empty() const noexcept { return 0 == size; }
00037
00039
00042     inline virtual ulong Size() const noexcept { return size; }
00043   };
00044
00048   class ClearableContainer : virtual public Container
00049   {
00050
00051   public:
00053     virtual ~ClearableContainer() = default;
00054
00056     ClearableContainer &operator=(const ClearableContainer &) = delete;
00058     ClearableContainer &operator=(ClearableContainer &&) noexcept = delete;
00059
00061     bool operator==(const ClearableContainer &Ccon) const noexcept = delete;
00062     bool operator!=(const ClearableContainer &Ccon) const noexcept = delete;
00063
00065     virtual void Clear() = 0;
00066   };
00067
00071   class ResizableContainer : virtual public ClearableContainer
00072   {
00073
00074   public:
00076     virtual ~ResizableContainer() = default;
00077
00079     ResizableContainer &operator=(const ResizableContainer &Ccon) noexcept = delete;
00080
00082     ResizableContainer &operator=(ResizableContainer &&Ccon) noexcept = delete;
00083
00084     bool operator==(const ResizableContainer &Ccon) const noexcept = delete;
```

```
00085     bool operator!=(const ResizableContainer &Ccon) const noexcept = delete;
00086
00088
00091     virtual void Resize(ulong) = 0;
00092
00093
00094     // Specific member function (inherited from ClearableContainer)
00095
00097     void Clear() override { Resize(0); }
00098   };
00099
00100 }
00101
00102 #endif
```

7.27 Exercise1/zlasdtest/container/container.hpp File Reference

```
#include "../../container/container.hpp"
```

Functions

- void Empty (uint &, uint &, const lasd::Container &, bool)
- void Size (uint &, uint &, const lasd::Container &, bool, ulong)

7.27.1 Function Documentation

7.27.1.1 Empty()

```
void Empty (
            uint & testnum,
            uint & testerr,
            const lasd::Container & con,
            bool chk)
```

Definition at line 12 of file container.cpp.

7.27.1.2 Size()

```
void Size (
            uint & testnum,
            uint & testerr,
            const lasd::Container & con,
            bool chk,
            ulong siz)
```

Definition at line 20 of file container.cpp.

7.28 container.hpp

Go to the documentation of this file.

```
00001
00002 #ifndef CONTAINERTEST_HPP
00003 #define CONTAINERTEST_HPP
00004
00005 #include "../../container/container.hpp"
00006
00007 /* ********************************************************************** */
00008
00009 // Container member functions!
00010
00011 void Empty(uint &, uint &, const lasd::Container &, bool);
00012
00013 void Size(uint &, uint &, const lasd::Container &, bool, ulong);
00014
00015 /* ********************************************************************** */
00016
00017 #endif
```

7.29 Exercise1/container/dictionary.hpp File Reference

```
#include "testable.hpp"
#include "mappable.hpp"
#include "dictionary.cpp"
```

Classes

- class lasd::DictionaryContainer< Data >

  La classe DictionaryContainer definisce al suo interno tutti quei metodi che permettono ad una struttura dati di funzionare come un dizionario.
- class lasd::OrderedDictionaryContainer< Data >

  Classe che estende DictionaryContainer con capacità di ordinamento.

Namespaces

- namespace lasd

7.30 dictionary.hpp

Go to the documentation of this file.

```
00001
00002 #ifndef DICTIONARY_HPP
00003 #define DICTIONARY_HPP
00004
00005 #include "testable.hpp"
00006 #include "mappable.hpp"
00007
00008 namespace lasd
00009 {
00010
00012   template <typename Data>
00013   class DictionaryContainer : virtual public TestableContainer<Data>
00014   {
00015
00016   public:
00018     virtual ~DictionaryContainer() = default;
00019
```

```
00021     DictionaryContainer &operator=(const DictionaryContainer &) = delete;
00022
00024     DictionaryContainer &operator=(DictionaryContainer &&) = delete;
00025
00026     // Comparison operators
00027     bool operator==(const DictionaryContainer &) const noexcept = delete;
00028     bool operator!=(const DictionaryContainer &) const noexcept = delete;
00029
00030     // Specific member functions
00031
00033
00037     virtual bool Insert(const Data &val) = 0;
00038
00040
00044     virtual bool Insert(Data &&val) = 0;
00045
00047
00051     virtual bool Remove(const Data &val) = 0;
00052
00054
00058     virtual inline bool InsertAll(const TraversableContainer<Data> &container);
00059
00061
00065     virtual inline bool InsertAll(MappableContainer<Data> &&container);
00066
00068
00072     virtual inline bool RemoveAll(const TraversableContainer<Data> &container);
00073
00075
00079     virtual inline bool InsertSome(const TraversableContainer<Data> &container);
00080
00082
00086     virtual inline bool InsertSome(MappableContainer<Data> &&container);
00087
00089
00093     virtual inline bool RemoveSome(const TraversableContainer<Data> &container);
00094   };
00095
00097
00100   template <typename Data>
00101   class OrderedDictionaryContainer : virtual public DictionaryContainer<Data>
00102   {
00103
00104   public:
00106     virtual ~OrderedDictionaryContainer() = default;
00107
00109     OrderedDictionaryContainer &operator=(const OrderedDictionaryContainer &) = delete;
00111     OrderedDictionaryContainer &operator=(OrderedDictionaryContainer &&) = delete;
00112
00113     bool operator==(const OrderedDictionaryContainer &) const noexcept = delete;
00114     bool operator!=(const OrderedDictionaryContainer &) const noexcept = delete;
00115
00116     // Specific member functions
00117
00119
00123     virtual const Data &Min() const = 0;
00124
00126
00130     virtual Data MinNRemove() = 0;
00131
00133
00136     virtual void RemoveMin() = 0;
00137
00139
00143     virtual const Data &Max() const = 0;
00144
00146
00150     virtual Data MaxNRemove() = 0;
00151
00153
00156     virtual void RemoveMax() = 0;
00157
00159
00164     virtual const Data &Predecessor(const Data &val) const = 0;
00165
00167
00172     virtual Data PredecessorNRemove(const Data &val) = 0;
00173
00175
00179     virtual void RemovePredecessor(const Data &val) = 0;
00180
00182
00187     virtual const Data &Successor(const Data &val) const = 0;
00188
00190
00195     virtual Data SuccessorNRemove(const Data &val) = 0;
00196
```

```
00198
00202     virtual void RemoveSuccessor(const Data &val) = 0;
00203   };
00204
00205 }
00206
00207 #include "dictionary.cpp"
00208
00209 #endif
```

7.31 Exercise1/zlasdtest/container/dictionary.hpp File Reference

```
#include "../../container/dictionary.hpp"
```

Functions

- template<typename Data>
  void InsertC (uint &testnum, uint &testerr, lasd::DictionaryContainer< Data > &con, const Data &val)
- template<typename Data>
  void InsertM (uint &testnum, uint &testerr, lasd::DictionaryContainer< Data > &con, Data &&val)
- template<typename Data>
  void Remove (uint &testnum, uint &testerr, lasd::DictionaryContainer< Data > &con, const Data &val)
- template<typename Data>
  void InsertC (uint &testnum, uint &testerr, lasd::DictionaryContainer< Data > &con, bool chk, const Data &val)
- template<typename Data>
  void InsertM (uint &testnum, uint &testerr, lasd::DictionaryContainer< Data > &con, bool chk, Data &&val)
- template<typename Data>
  void Remove (uint &testnum, uint &testerr, lasd::DictionaryContainer< Data > &con, bool chk, const Data &val)
- template<typename Data>
  void InsertC (uint &testnum, uint &testerr, lasd::DictionaryContainer< Data > &con, const lasd::TraversableContainer< Data > &mc)
- template<typename Data>
  void InsertM (uint &testnum, uint &testerr, lasd::DictionaryContainer< Data > &con, lasd::MappableContainer< Data > &&mc)
- template<typename Data>
  void Remove (uint &testnum, uint &testerr, lasd::DictionaryContainer< Data > &con, const lasd::TraversableContainer< Data > &mc)
- template<typename Data>
  void InsertAllC (uint &testnum, uint &testerr, lasd::DictionaryContainer< Data > &con, bool chk, const lasd::TraversableContainer< Data > &mc)
- template<typename Data>
  void InsertAllM (uint &testnum, uint &testerr, lasd::DictionaryContainer< Data > &con, bool chk, lasd::MappableContainer< Data > &&mc)
- template<typename Data>
  void RemoveAll (uint &testnum, uint &testerr, lasd::DictionaryContainer< Data > &con, bool chk, const lasd::TraversableContainer< Data > &mc)
- template<typename Data>
  void InsertSomeC (uint &testnum, uint &testerr, lasd::DictionaryContainer< Data > &con, bool chk, const lasd::TraversableContainer< Data > &mc)
- template<typename Data>
  void InsertSomeM (uint &testnum, uint &testerr, lasd::DictionaryContainer< Data > &con, bool chk, lasd::MappableContainer< Data > &&mc)

- template<typename Data>
  void RemoveSome (uint &testnum, uint &testerr, lasd::DictionaryContainer< Data > &con, bool chk, const lasd::TraversableContainer< Data > &mc)
- template<typename Data>
  void Min (uint &testnum, uint &testerr, lasd::OrderedDictionaryContainer< Data > &con, bool chk, const Data &val)
- template<typename Data>
  void RemoveMin (uint &testnum, uint &testerr, lasd::OrderedDictionaryContainer< Data > &con, bool chk)
- template<typename Data>
  void MinNRemove (uint &testnum, uint &testerr, lasd::OrderedDictionaryContainer< Data > &con, bool chk, const Data &val)
- template<typename Data>
  void Max (uint &testnum, uint &testerr, lasd::OrderedDictionaryContainer< Data > &con, bool chk, const Data &val)
- template<typename Data>
  void RemoveMax (uint &testnum, uint &testerr, lasd::OrderedDictionaryContainer< Data > &con, bool chk)
- template<typename Data>
  void MaxNRemove (uint &testnum, uint &testerr, lasd::OrderedDictionaryContainer< Data > &con, bool chk, const Data &val)
- template<typename Data>
  void Predecessor (uint &testnum, uint &testerr, lasd::OrderedDictionaryContainer< Data > &con, bool chk, const Data &prd, const Data &val)
- template<typename Data>
  void RemovePredecessor (uint &testnum, uint &testerr, lasd::OrderedDictionaryContainer< Data > &con, bool chk, const Data &prd)
- template<typename Data>
  void PredecessorNRemove (uint &testnum, uint &testerr, lasd::OrderedDictionaryContainer< Data > &con, bool chk, const Data &prd, const Data &val)
- template<typename Data>
  void Successor (uint &testnum, uint &testerr, lasd::OrderedDictionaryContainer< Data > &con, bool chk, const Data &prd, const Data &val)
- template<typename Data>
  void RemoveSuccessor (uint &testnum, uint &testerr, lasd::OrderedDictionaryContainer< Data > &con, bool chk, const Data &prd)
- template<typename Data>
  void SuccessorNRemove (uint &testnum, uint &testerr, lasd::OrderedDictionaryContainer< Data > &con, bool chk, const Data &prd, const Data &val)

7.31.1 Function Documentation

7.31.1.1 InsertAllC()

```
template<typename Data>
void InsertAllC (
            uint & testnum,
            uint & testerr,
            lasd::DictionaryContainer< Data > & con,
            bool chk,
            const lasd::TraversableContainer< Data > & mc)
```

Definition at line 157 of file dictionary.hpp.

7.31.1.2 InsertAllM()

```
template<typename Data>
void InsertAllM (
            uint & testnum,
            uint & testerr,
            lasd::DictionaryContainer< Data > & con,
            bool chk,
            lasd::MappableContainer< Data > && mc)
```

Definition at line 173 of file dictionary.hpp.

7.31.1.3 InsertC() [1/3]

```
template<typename Data>
void InsertC (
            uint & testnum,
            uint & testerr,
            lasd::DictionaryContainer< Data > & con,
            bool chk,
            const Data & val)
```

Definition at line 60 of file dictionary.hpp.

7.31.1.4 InsertC() [2/3]

```
template<typename Data>
void InsertC (
            uint & testnum,
            uint & testerr,
            lasd::DictionaryContainer< Data > & con,
            const Data & val)
```

Definition at line 12 of file dictionary.hpp.

7.31.1.5 InsertC() [3/3]

```
template<typename Data>
void InsertC (
            uint & testnum,
            uint & testerr,
            lasd::DictionaryContainer< Data > & con,
            const lasd::TraversableContainer< Data > & mc)
```

Definition at line 109 of file dictionary.hpp.

7.31.1.6 InsertM() [1/3]

```
template<typename Data>
void InsertM (
            uint & testnum,
            uint & testerr,
            lasd::DictionaryContainer< Data > & con,
            bool chk,
            Data && val)
```

Definition at line 76 of file dictionary.hpp.

7.31.1.7 InsertM() [2/3]

```
template<typename Data>
void InsertM (
             uint & testnum,
             uint & testerr,
             lasd::DictionaryContainer< Data > & con,
             Data && val)
```

Definition at line 28 of file dictionary.hpp.

7.31.1.8 InsertM() [3/3]

```
template<typename Data>
void InsertM (
             uint & testnum,
             uint & testerr,
             lasd::DictionaryContainer< Data > & con,
             lasd::MappableContainer< Data > && mc)
```

Definition at line 125 of file dictionary.hpp.

7.31.1.9 InsertSomeC()

```
template<typename Data>
void InsertSomeC (
             uint & testnum,
             uint & testerr,
             lasd::DictionaryContainer< Data > & con,
             bool chk,
             const lasd::TraversableContainer< Data > & mc)
```

Definition at line 205 of file dictionary.hpp.

7.31.1.10 InsertSomeM()

```
template<typename Data>
void InsertSomeM (
             uint & testnum,
             uint & testerr,
             lasd::DictionaryContainer< Data > & con,
             bool chk,
             lasd::MappableContainer< Data > && mc)
```

Definition at line 221 of file dictionary.hpp.

7.31.1.11 Max()

```
template<typename Data>
void Max (
             uint & testnum,
             uint & testerr,
             lasd::OrderedDictionaryContainer< Data > & con,
             bool chk,
             const Data & val)
```

Definition at line 312 of file dictionary.hpp.

7.31.1.12 MaxNRemove()

```
template<typename Data>
void MaxNRemove (
             uint & testnum,
             uint & testerr,
             lasd::OrderedDictionaryContainer< Data > & con,
             bool chk,
             const Data & val)
```

Definition at line 349 of file dictionary.hpp.

7.31.1.13 Min()

```
template<typename Data>
void Min (
             uint & testnum,
             uint & testerr,
             lasd::OrderedDictionaryContainer< Data > & con,
             bool chk,
             const Data & val)
```

Definition at line 257 of file dictionary.hpp.

7.31.1.14 MinNRemove()

```
template<typename Data>
void MinNRemove (
             uint & testnum,
             uint & testerr,
             lasd::OrderedDictionaryContainer< Data > & con,
             bool chk,
             const Data & val)
```

Definition at line 294 of file dictionary.hpp.

7.31.1.15 Predecessor()

```
template<typename Data>
void Predecessor (
             uint & testnum,
             uint & testerr,
             lasd::OrderedDictionaryContainer< Data > & con,
             bool chk,
             const Data & prd,
             const Data & val)
```

Definition at line 367 of file dictionary.hpp.

7.31.1.16 PredecessorNRemove()

```
template<typename Data>
void PredecessorNRemove (
              uint & testnum,
              uint & testerr,
              lasd::OrderedDictionaryContainer< Data > & con,
              bool chk,
              const Data & prd,
              const Data & val)
```

Definition at line 404 of file dictionary.hpp.

7.31.1.17 Remove() [1/3]

```
template<typename Data>
void Remove (
              uint & testnum,
              uint & testerr,
              lasd::DictionaryContainer< Data > & con,
              bool chk,
              const Data & val)
```

Definition at line 93 of file dictionary.hpp.

7.31.1.18 Remove() [2/3]

```
template<typename Data>
void Remove (
              uint & testnum,
              uint & testerr,
              lasd::DictionaryContainer< Data > & con,
              const Data & val)
```

Definition at line 44 of file dictionary.hpp.

7.31.1.19 Remove() [3/3]

```
template<typename Data>
void Remove (
              uint & testnum,
              uint & testerr,
              lasd::DictionaryContainer< Data > & con,
              const lasd::TraversableContainer< Data > & mc)
```

Definition at line 141 of file dictionary.hpp.

7.31.1.20 RemoveAll()

```
template<typename Data>
void RemoveAll (
             uint & testnum,
             uint & testerr,
             lasd::DictionaryContainer< Data > & con,
             bool chk,
             const lasd::TraversableContainer< Data > & mc)
```

Definition at line 189 of file dictionary.hpp.

7.31.1.21 RemoveMax()

```
template<typename Data>
void RemoveMax (
             uint & testnum,
             uint & testerr,
             lasd::OrderedDictionaryContainer< Data > & con,
             bool chk)
```

Definition at line 330 of file dictionary.hpp.

7.31.1.22 RemoveMin()

```
template<typename Data>
void RemoveMin (
             uint & testnum,
             uint & testerr,
             lasd::OrderedDictionaryContainer< Data > & con,
             bool chk)
```

Definition at line 275 of file dictionary.hpp.

7.31.1.23 RemovePredecessor()

```
template<typename Data>
void RemovePredecessor (
             uint & testnum,
             uint & testerr,
             lasd::OrderedDictionaryContainer< Data > & con,
             bool chk,
             const Data & prd)
```

Definition at line 385 of file dictionary.hpp.

7.31.1.24 RemoveSome()

```
template<typename Data>
void RemoveSome (
             uint & testnum,
             uint & testerr,
             lasd::DictionaryContainer< Data > & con,
             bool chk,
             const lasd::TraversableContainer< Data > & mc)
```

Definition at line 237 of file dictionary.hpp.

7.31.1.25 RemoveSuccessor()

```
template<typename Data>
void RemoveSuccessor (
             uint & testnum,
             uint & testerr,
             lasd::OrderedDictionaryContainer< Data > & con,
             bool chk,
             const Data & prd)
```

Definition at line 440 of file dictionary.hpp.

7.31.1.26 Successor()

```
template<typename Data>
void Successor (
             uint & testnum,
             uint & testerr,
             lasd::OrderedDictionaryContainer< Data > & con,
             bool chk,
             const Data & prd,
             const Data & val)
```

Definition at line 422 of file dictionary.hpp.

7.31.1.27 SuccessorNRemove()

```
template<typename Data>
void SuccessorNRemove (
             uint & testnum,
             uint & testerr,
             lasd::OrderedDictionaryContainer< Data > & con,
             bool chk,
             const Data & prd,
             const Data & val)
```

Definition at line 459 of file dictionary.hpp.

7.32 dictionary.hpp

Go to the documentation of this file.

```
00001
00002 #ifndef DICTIONARYTEST_HPP
00003 #define DICTIONARYTEST_HPP
00004
00005 #include "../../container/dictionary.hpp"
00006
00007 /* ************************************************************************** */
00008
00009 // DictionaryContainer member functions!
00010
00011 template <typename Data>
00012 void InsertC(uint & testnum, uint & testerr, lasd::DictionaryContainer<Data> & con, const Data & val)
      {
00013   testnum++;
00014   bool tst = true;
00015   try {
```

```
00016     std::cout « " " « testnum « " (" « testerr « ") Insertion in the dictionary container of the value
      \"" « val « "\": ";
00017     con.Insert(val);
00018     std::cout « "Correct!" « std::endl;
00019   }
00020   catch (std::exception & exc) {
00021     tst = false;
00022     std::cout « "\"" « exc.what() « "\": " « "Error!" « std::endl;
00023   }
00024   testerr += (1 - (uint) tst);
00025 }
00026
00027 template <typename Data>
00028 void InsertM(uint & testnum, uint & testerr, lasd::DictionaryContainer<Data> & con, Data && val) {
00029   testnum++;
00030   bool tst = true;
00031   try {
00032     std::cout « " " « testnum « " (" « testerr « ") Insertion in the dictionary container of the value
      \"" « val « "\": ";
00033     con.Insert(std::move(val));
00034     std::cout « "Correct!" « std::endl;
00035   }
00036   catch (std::exception & exc) {
00037     tst = false;
00038     std::cout « "\"" « exc.what() « "\": " « "Error!" « std::endl;
00039   }
00040   testerr += (1 - (uint) tst);
00041 }
00042
00043 template <typename Data>
00044 void Remove(uint & testnum, uint & testerr, lasd::DictionaryContainer<Data> & con, const Data & val) {
00045   testnum++;
00046   bool tst = true;
00047   try {
00048     std::cout « " " « testnum « " (" « testerr « ") Removal from the dictionary container of the value
      \"" « val « "\": ";
00049     con.Remove(val);
00050     std::cout « "Correct!" « std::endl;
00051   }
00052   catch (std::exception & exc) {
00053     tst = false;
00054     std::cout « "\"" « exc.what() « "\": " « "Error!" « std::endl;
00055   }
00056   testerr += (1 - (uint) tst);
00057 }
00058
00059 template <typename Data>
00060 void InsertC(uint & testnum, uint & testerr, lasd::DictionaryContainer<Data> & con, bool chk, const
      Data & val) {
00061   testnum++;
00062   bool tst;
00063   try {
00064     std::cout « " " « testnum « " (" « testerr « ") Insertion in the dictionary container of the value
      \"" « val « "\"; ";
00065     std::cout « "it " « ((tst = con.Insert(val)) ? "has" : "has not") « " been inserted: ";
00066     std::cout « ((tst = (tst == chk)) ? "Correct" : "Error") « "!" « std::endl;
00067   }
00068   catch (std::exception & exc) {
00069     tst = false;
00070     std::cout « "\"" « exc.what() « "\": " « "Error!" « std::endl;
00071   }
00072   testerr += (1 - (uint) tst);
00073 }
00074
00075 template <typename Data>
00076 void InsertM(uint & testnum, uint & testerr, lasd::DictionaryContainer<Data> & con, bool chk, Data &&
      val) {
00077   testnum++;
00078   bool tst;
00079   try {
00080     std::cout « " " « testnum « " (" « testerr « ") Insertion in the dictionary container of the value
      \"" « val « "\"; ";
00081     std::cout « "it " « ((tst = con.Insert(std::move(val))) ? "has" : "has not") « " been inserted: ";
00082     std::cout « ((tst = (tst == chk)) ? "Correct" : "Error") « "!" « std::endl;
00083     std::cout « "Correct!" « std::endl;
00084   }
00085   catch (std::exception & exc) {
00086     tst = false;
00087     std::cout « "\"" « exc.what() « "\": " « "Error!" « std::endl;
00088   }
00089   testerr += (1 - (uint) tst);
00090 }
00091
00092 template <typename Data>
00093 void Remove(uint & testnum, uint & testerr, lasd::DictionaryContainer<Data> & con, bool chk, const
      Data & val) {
00094   testnum++;
```

```
00095   bool tst;
00096   try {
00097     std::cout « " " « testnum « " (" « testerr « ") Removal from the dictionary container of the value
      \"" « val « "\"; ";
00098     std::cout « "it " « ((tst = con.Remove(val)) ? "has" : "has not") « " been removed: ";
00099     std::cout « ((tst = (tst == chk)) ? "Correct" : "Error") « "!" « std::endl;
00100   }
00101   catch (std::exception & exc) {
00102     tst = false;
00103     std::cout « "\"" « exc.what() « "\": " « "Error!" « std::endl;
00104   }
00105   testerr += (1 - (uint) tst);
00106 }
00107
00108 template <typename Data>
00109 void InsertC(uint & testnum, uint & testerr, lasd::DictionaryContainer<Data> & con, const
      lasd::TraversableContainer<Data> & mc) {
00110   testnum++;
00111   bool tst = true;
00112   try {
00113     std::cout « " " « testnum « " (" « testerr « ") Insertion in the dictionary container of the
      values of the given linear container: ";
00114     con.InsertAll(mc);
00115     std::cout « "Correct!" « std::endl;
00116   }
00117   catch (std::exception & exc) {
00118     tst = false;
00119     std::cout « "\"" « exc.what() « "\": " « "Error!" « std::endl;
00120   }
00121   testerr += (1 - (uint) tst);
00122 }
00123
00124 template <typename Data>
00125 void InsertM(uint & testnum, uint & testerr, lasd::DictionaryContainer<Data> & con,
      lasd::MappableContainer<Data> && mc) {
00126   testnum++;
00127   bool tst = true;
00128   try {
00129     std::cout « " " « testnum « " (" « testerr « ") Insertion in the dictionary container of the
      values of the given linear container: ";
00130     con.InsertAll(std::move(mc));
00131     std::cout « "Correct!" « std::endl;
00132   }
00133   catch (std::exception & exc) {
00134     tst = false;
00135     std::cout « "\"" « exc.what() « "\": " « "Error!" « std::endl;
00136   }
00137   testerr += (1 - (uint) tst);
00138 }
00139
00140 template <typename Data>
00141 void Remove(uint & testnum, uint & testerr, lasd::DictionaryContainer<Data> & con, const
      lasd::TraversableContainer<Data> & mc) {
00142   testnum++;
00143   bool tst = true;
00144   try {
00145     std::cout « " " « testnum « " (" « testerr « ") Removal from the dictionary container of the
      values of the given linear container: ";
00146     con.RemoveAll(mc);
00147     std::cout « "Correct!" « std::endl;
00148   }
00149   catch (std::exception & exc) {
00150     tst = false;
00151     std::cout « "\"" « exc.what() « "\": " « "Error!" « std::endl;
00152   }
00153   testerr += (1 - (uint) tst);
00154 }
00155
00156 template <typename Data>
00157 void InsertAllC(uint & testnum, uint & testerr, lasd::DictionaryContainer<Data> & con, bool chk, const
      lasd::TraversableContainer<Data> & mc) {
00158   testnum++;
00159   bool tst = true;
00160   try {
00161     std::cout « " " « testnum « " (" « testerr « ") Insertion in the dictionary container of the
      values of the given mappable container; ";
00162     std::cout « ((tst = con.InsertAll(mc)) ? "all" : "not all") « " values have been inserted: ";
00163     std::cout « ((tst = (tst == chk)) ? "Correct" : "Error") « "!" « std::endl;
00164   }
00165   catch (std::exception & exc) {
00166     tst = false;
00167     std::cout « "\"" « exc.what() « "\": " « "Error!" « std::endl;
00168   }
00169   testerr += (1 - (uint) tst);
00170 }
00171
00172 template <typename Data>
```

```
00173 void InsertAllM(uint & testnum, uint & testerr, lasd::DictionaryContainer<Data> & con, bool chk,
      lasd::MappableContainer<Data> && mc) {
00174   testnum++;
00175   bool tst = true;
00176   try {
00177     std::cout « " " « testnum « " (" « testerr « ") Insertion in the dictionary container of the
      values of the given mappable container; ";
00178     std::cout « ((tst = con.InsertAll(std::move(mc))) ? "all" : "not all") « " values have been
      inserted: ";
00179     std::cout « ((tst = (tst == chk)) ? "Correct" : "Error") « "!" « std::endl;
00180   }
00181   catch (std::exception & exc) {
00182     tst = false;
00183     std::cout « "\"" « exc.what() « "\": " « "Error!" « std::endl;
00184   }
00185   testerr += (1 - (uint) tst);
00186 }
00187
00188 template <typename Data>
00189 void RemoveAll(uint & testnum, uint & testerr, lasd::DictionaryContainer<Data> & con, bool chk, const
      lasd::TraversableContainer<Data> & mc) {
00190   testnum++;
00191   bool tst = true;
00192   try {
00193     std::cout « " " « testnum « " (" « testerr « ") Removal from the dictionary container of the
      values of the given mappable container; ";
00194     std::cout « ((tst = con.RemoveAll(mc)) ? "all" : "not all") « " values have been removed: ";
00195     std::cout « ((tst = (tst == chk)) ? "Correct" : "Error") « "!" « std::endl;
00196   }
00197   catch (std::exception & exc) {
00198     tst = false;
00199     std::cout « "\"" « exc.what() « "\": " « "Error!" « std::endl;
00200   }
00201   testerr += (1 - (uint) tst);
00202 }
00203
00204 template <typename Data>
00205 void InsertSomeC(uint & testnum, uint & testerr, lasd::DictionaryContainer<Data> & con, bool chk,
      const lasd::TraversableContainer<Data> & mc) {
00206   testnum++;
00207   bool tst = true;
00208   try {
00209     std::cout « " " « testnum « " (" « testerr « ") Insertion in the dictionary container of the
      values of the given mappable container; ";
00210     std::cout « ((tst = con.InsertSome(mc)) ? "some value" : "none of the values") « " has been
      inserted: ";
00211     std::cout « ((tst = (tst == chk)) ? "Correct" : "Error") « "!" « std::endl;
00212   }
00213   catch (std::exception & exc) {
00214     tst = false;
00215     std::cout « "\"" « exc.what() « "\": " « "Error!" « std::endl;
00216   }
00217   testerr += (1 - (uint) tst);
00218 }
00219
00220 template <typename Data>
00221 void InsertSomeM(uint & testnum, uint & testerr, lasd::DictionaryContainer<Data> & con, bool chk,
      lasd::MappableContainer<Data> && mc) {
00222   testnum++;
00223   bool tst = true;
00224   try {
00225     std::cout « " " « testnum « " (" « testerr « ") Insertion in the dictionary container of the
      values of the given mappable container; ";
00226     std::cout « ((tst = con.InsertSome(std::move(mc))) ? "some value" : "none of the values") « " has
      been inserted: ";
00227     std::cout « ((tst = (tst == chk)) ? "Correct" : "Error") « "!" « std::endl;
00228   }
00229   catch (std::exception & exc) {
00230     tst = false;
00231     std::cout « "\"" « exc.what() « "\": " « "Error!" « std::endl;
00232   }
00233   testerr += (1 - (uint) tst);
00234 }
00235
00236 template <typename Data>
00237 void RemoveSome(uint & testnum, uint & testerr, lasd::DictionaryContainer<Data> & con, bool chk, const
      lasd::TraversableContainer<Data> & mc) {
00238   testnum++;
00239   bool tst = true;
00240   try {
00241     std::cout « " " « testnum « " (" « testerr « ") Removal from the dictionary container of the
      values of the given mappable container; ";
00242     std::cout « ((tst = con.RemoveSome(mc)) ? "some value" : "none of the values") « " has been
      removed: ";
00243     std::cout « ((tst = (tst == chk)) ? "Correct" : "Error") « "!" « std::endl;
00244   }
00245   catch (std::exception & exc) {
```

```
00246     tst = false;
00247     std::cout « "\"" « exc.what() « "\": " « "Error!" « std::endl;
00248   }
00249   testerr += (1 - (uint) tst);
00250 }
00251
00252 /* ********************************************************************** */
00253
00254 // OrderedDictionaryContainer member functions!
00255
00256 template <typename Data>
00257 void Min(uint & testnum, uint & testerr, lasd::OrderedDictionaryContainer<Data> & con, bool chk, const
      Data & val) {
00258   bool tst;
00259   testnum++;
00260   try {
00261     std::cout « " " « testnum « " (" « testerr « ") Min of the ordered dictionary container with value
      \"" « con.Min() « "\": ";
00262     std::cout « ((tst = ((con.Min() == val) == chk)) ? "Correct" : "Error") « "!" « std::endl;
00263   }
00264   catch (std::length_error & exc) {
00265     std::cout « "\"" « exc.what() « "\": " « ((tst = !chk) ? "Correct" : "Error") « "!" « std::endl;
00266   }
00267   catch (std::exception & exc) {
00268     tst = false;
00269     std::cout « std::endl « "Wrong exception: " « exc.what() « "!" « std::endl;
00270   }
00271   testerr += (1 - (uint) tst);
00272 }
00273
00274 template <typename Data>
00275 void RemoveMin(uint & testnum, uint & testerr, lasd::OrderedDictionaryContainer<Data> & con, bool chk)
      {
00276   bool tst;
00277   testnum++;
00278   try {
00279     std::cout « " " « testnum « " (" « testerr « ") Remove min from the the ordered dictionary
      container: ";
00280     con.RemoveMin();
00281     std::cout « ((tst = chk) ? "Correct" : "Error") « "!" « std::endl;
00282   }
00283   catch (std::length_error & exc) {
00284     std::cout « "\"" « exc.what() « "\": " « ((tst = !chk) ? "Correct" : "Error") « "!" « std::endl;
00285   }
00286   catch (std::exception & exc) {
00287     tst = false;
00288     std::cout « std::endl « "Wrong exception: " « exc.what() « "!" « std::endl;
00289   }
00290   testerr += (1 - (uint) tst);
00291 }
00292
00293 template <typename Data>
00294 void MinNRemove(uint & testnum, uint & testerr, lasd::OrderedDictionaryContainer<Data> & con, bool
      chk, const Data & val) {
00295   bool tst;
00296   testnum++;
00297   try {
00298     std::cout « " " « testnum « " (" « testerr « ") MinNRemove from the ordered dictionary container
      with value \"" « con.Min() « "\": ";
00299     std::cout « ((tst = ((con.MinNRemove() == val) == chk)) ? "Correct" : "Error") « "!" « std::endl;
00300   }
00301   catch (std::length_error & exc) {
00302     std::cout « "\"" « exc.what() « "\": " « ((tst = !chk) ? "Correct" : "Error") « "!" « std::endl;
00303   }
00304   catch (std::exception & exc) {
00305     tst = false;
00306     std::cout « std::endl « "Wrong exception: " « exc.what() « "!" « std::endl;
00307   }
00308   testerr += (1 - (uint) tst);
00309 }
00310
00311 template <typename Data>
00312 void Max(uint & testnum, uint & testerr, lasd::OrderedDictionaryContainer<Data> & con, bool chk, const
      Data & val) {
00313   bool tst;
00314   testnum++;
00315   try {
00316     std::cout « " " « testnum « " (" « testerr « ") Max of the ordered dictionary container with value
      \"" « con.Max() « "\": ";
00317     std::cout « ((tst = ((con.Max() == val) == chk)) ? "Correct" : "Error") « "!" « std::endl;
00318   }
00319   catch (std::length_error & exc) {
00320     std::cout « "\"" « exc.what() « "\": " « ((tst = !chk) ? "Correct" : "Error") « "!" « std::endl;
00321   }
00322   catch (std::exception & exc) {
00323     tst = false;
00324     std::cout « std::endl « "Wrong exception: " « exc.what() « "!" « std::endl;
```

```
00325   }
00326   testerr += (1 - (uint) tst);
00327 }
00328
00329 template <typename Data>
00330 void RemoveMax(uint & testnum, uint & testerr, lasd::OrderedDictionaryContainer<Data> & con, bool chk)
    {
00331   bool tst;
00332   testnum++;
00333   try {
00334     std::cout « " " « testnum « " (" « testerr « ") Remove max from the ordered dictionary container:
    ";
00335     con.RemoveMax();
00336     std::cout « ((tst = chk) ? "Correct" : "Error") « "!" « std::endl;
00337   }
00338   catch (std::length_error & exc) {
00339     std::cout « "\"" « exc.what() « "\": " « ((tst = !chk) ? "Correct" : "Error") « "!" « std::endl;
00340   }
00341   catch (std::exception & exc) {
00342     tst = false;
00343     std::cout « std::endl « "Wrong exception: " « exc.what() « "!" « std::endl;
00344   }
00345   testerr += (1 - (uint) tst);
00346 }
00347
00348 template <typename Data>
00349 void MaxNRemove(uint & testnum, uint & testerr, lasd::OrderedDictionaryContainer<Data> & con, bool
    chk, const Data & val) {
00350   bool tst;
00351   testnum++;
00352   try {
00353     std::cout « " " « testnum « " (" « testerr « ") MaxNRemove from the ordered dictionary container
    with value \"" « con.Max() « "\": ";
00354     std::cout « ((tst = ((con.MaxNRemove() == val) == chk)) ? "Correct" : "Error") « "!" « std::endl;
00355   }
00356   catch (std::length_error & exc) {
00357     std::cout « "\"" « exc.what() « "\": " « ((tst = !chk) ? "Correct" : "Error") « "!" « std::endl;
00358   }
00359   catch (std::exception & exc) {
00360     tst = false;
00361     std::cout « std::endl « "Wrong exception: " « exc.what() « "!" « std::endl;
00362   }
00363   testerr += (1 - (uint) tst);
00364 }
00365
00366 template <typename Data>
00367 void Predecessor(uint & testnum, uint & testerr, lasd::OrderedDictionaryContainer<Data> & con, bool
    chk, const Data & prd, const Data & val) {
00368   bool tst;
00369   testnum++;
00370   try {
00371     std::cout « " " « testnum « " (" « testerr « ") Predecessor of " « prd « " with value \"" «
    con.Predecessor(prd) « "\": ";
00372     std::cout « ((tst = ((con.Predecessor(prd) == val) == chk)) ? "Correct" : "Error") « "!" «
    std::endl;
00373   }
00374   catch (std::length_error & exc) {
00375     std::cout « "\"" « exc.what() « "\": " « ((tst = !chk) ? "Correct" : "Error") « "!" « std::endl;
00376   }
00377   catch (std::exception & exc) {
00378     tst = false;
00379     std::cout « std::endl « "Wrong exception: " « exc.what() « "!" « std::endl;
00380   }
00381   testerr += (1 - (uint) tst);
00382 }
00383
00384 template <typename Data>
00385 void RemovePredecessor(uint & testnum, uint & testerr, lasd::OrderedDictionaryContainer<Data> & con,
    bool chk, const Data & prd) {
00386   bool tst;
00387   testnum++;
00388   try {
00389     std::cout « " " « testnum « " (" « testerr « ") Remove predecessor of " « prd « " from the the
    ordered dictionary container: \"" « con.Predecessor(prd) « "\": ";
00390     con.RemovePredecessor(prd);
00391     std::cout « ((tst = chk) ? "Correct" : "Error") « "!" « std::endl;
00392   }
00393   catch (std::length_error & exc) {
00394     std::cout « "\"" « exc.what() « "\": " « ((tst = !chk) ? "Correct" : "Error") « "!" « std::endl;
00395   }
00396   catch (std::exception & exc) {
00397     tst = false;
00398     std::cout « std::endl « "Wrong exception: " « exc.what() « "!" « std::endl;
00399   }
00400   testerr += (1 - (uint) tst);
00401 }
00402
```

```
00403 template <typename Data>
00404 void PredecessorNRemove(uint & testnum, uint & testerr, lasd::OrderedDictionaryContainer<Data> & con,
      bool chk, const Data & prd, const Data & val) {
00405   bool tst;
00406   testnum++;
00407   try {
00408     std::cout « " " « testnum « " (" « testerr « ") Remove predecessor of " « prd « " from the the
      ordered dictionary container: \"" « con.Predecessor(prd) « "\": ";
00409     std::cout « ((tst = ((con.PredecessorNRemove(prd) == val) == chk)) ? "Correct" : "Error") « "!" «
      std::endl;
00410   }
00411   catch (std::length_error & exc) {
00412     std::cout « "\"" « exc.what() « "\": " « ((tst = !chk) ? "Correct" : "Error") « "!" « std::endl;
00413   }
00414   catch (std::exception & exc) {
00415     tst = false;
00416     std::cout « std::endl « "Wrong exception: " « exc.what() « "!" « std::endl;
00417   }
00418   testerr += (1 - (uint) tst);
00419 }
00420
00421 template <typename Data>
00422 void Successor(uint & testnum, uint & testerr, lasd::OrderedDictionaryContainer<Data> & con, bool chk,
      const Data & prd, const Data & val) {
00423   bool tst;
00424   testnum++;
00425   try {
00426     std::cout « " " « testnum « " (" « testerr « ") Successor of " « prd « " with value \"" «
      con.Successor(prd) « "\": ";
00427     std::cout « ((tst = ((con.Successor(prd) == val) == chk)) ? "Correct" : "Error") « "!" «
      std::endl;
00428   }
00429   catch (std::length_error & exc) {
00430     std::cout « "\"" « exc.what() « "\": " « ((tst = !chk) ? "Correct" : "Error") « "!" « std::endl;
00431   }
00432   catch (std::exception & exc) {
00433     tst = false;
00434     std::cout « std::endl « "Wrong exception: " « exc.what() « "!" « std::endl;
00435   }
00436   testerr += (1 - (uint) tst);
00437 }
00438
00439 template <typename Data>
00440 void RemoveSuccessor(uint & testnum, uint & testerr, lasd::OrderedDictionaryContainer<Data> & con,
      bool chk, const Data & prd) {
00441   bool tst;
00442   testnum++;
00443   try {
00444     std::cout « " " « testnum « " (" « testerr « ") Remove successor of " « prd « " from the the
      ordered dictionary container: \"" « con.Successor(prd) « "\": ";
00445     con.RemoveSuccessor(prd);
00446     std::cout « ((tst = chk) ? "Correct" : "Error") « "!" « std::endl;
00447   }
00448   catch (std::length_error & exc) {
00449     std::cout « "\"" « exc.what() « "\": " « ((tst = !chk) ? "Correct" : "Error") « "!" « std::endl;
00450   }
00451   catch (std::exception & exc) {
00452     tst = false;
00453     std::cout « std::endl « "Wrong exception: " « exc.what() « "!" « std::endl;
00454   }
00455   testerr += (1 - (uint) tst);
00456 }
00457
00458 template <typename Data>
00459 void SuccessorNRemove(uint & testnum, uint & testerr, lasd::OrderedDictionaryContainer<Data> & con,
      bool chk, const Data & prd, const Data & val) {
00460   bool tst;
00461   testnum++;
00462   try {
00463     std::cout « " " « testnum « " (" « testerr « ") Remove successor of " « prd « " from the the
      ordered dictionary container: \"" « con.Successor(prd) « "\": ";
00464     std::cout « ((tst = ((con.SuccessorNRemove(prd) == val) == chk)) ? "Correct" : "Error") « "!" «
      std::endl;
00465   }
00466   catch (std::length_error & exc) {
00467     std::cout « "\"" « exc.what() « "\": " « ((tst = !chk) ? "Correct" : "Error") « "!" « std::endl;
00468   }
00469   catch (std::exception & exc) {
00470     tst = false;
00471     std::cout « std::endl « "Wrong exception: " « exc.what() « "!" « std::endl;
00472   }
00473   testerr += (1 - (uint) tst);
00474 }
00475
00476 /* ********************************************************************** */
00477
00478 #endif
```

7.33 Exercise1/container/linear.hpp File Reference

```
#include "mappable.hpp"
#include "linear.cpp"
```

Classes

- class lasd::LinearContainer< Data >

  Classe astratta che rappresenta un contenitore lineare accessibile per posizione.
- class lasd::MutableLinearContainer< Data >
- class lasd::SortableLinearContainer< Data >

  Classe astratta che rappresenta un contenitore lineare ordinabile.

Namespaces

- namespace lasd

7.34 linear.hpp

Go to the documentation of this file.

```
00001 #ifndef LINEAR_HPP
00002 #define LINEAR_HPP
00003
00004 #include "mappable.hpp"
00005
00006 namespace lasd
00007 {
00008
00016   template <typename Data>
00017   class LinearContainer : virtual public PreOrderMappableContainer<Data>,
00018                           virtual public PostOrderMappableContainer<Data>
00019   {
00020
00021   protected:
00022     using Container::size;
00023
00024   public:
00026     virtual ~LinearContainer() = default;
00027
00028     // Copy assignment
00029     LinearContainer &operator=(const LinearContainer &) = delete;
00030
00031     // Move assignment
00032     LinearContainer &operator=(LinearContainer &&) noexcept = delete;
00033
00035     inline bool operator==(const LinearContainer &) const noexcept;
00036
00038     inline bool operator!=(const LinearContainer &) const noexcept;
00039
00046     virtual const Data &operator[](const ulong int index) const = 0;
00047
00054     virtual Data &operator[](const ulong int index) = 0;
00055
00061     virtual inline const Data &Front() const;
00062
00068     virtual inline Data &Front();
00069
00075     virtual inline const Data &Back() const;
00076
00082     virtual inline Data &Back();
00083
00085     inline void Traverse(TraverseFun) const override;
00086
00088     inline void PreOrderTraverse(TraverseFun) const override;
00089
00091     inline void PostOrderTraverse(TraverseFun) const override;
```

```
00092
00094     inline void Map(MapFun) override;
00095
00097     inline void PreOrderMap(MapFun) override;
00098
00100     inline void PostOrderMap(MapFun) override;
00101   };
00102
00103   template <typename Data>
00104   class MutableLinearContainer : virtual public LinearContainer<Data>,
      PreOrderMappableContainer<Data>, PostOrderMappableContainer<Data>
00105   {
00106
00107   public:
00111     virtual ~MutableLinearContainer() = default;
00112
00113     // Copy assignment
00114     MutableLinearContainer &operator=(const MutableLinearContainer &) = delete;
00115
00116     // Move assignment
00117     MutableLinearContainer &operator=(MutableLinearContainer &&) noexcept = delete;
00118
00126     virtual Data &operator[](const ulong index) = 0;
00127
00134     virtual Data &Front() = 0;
00135
00142     virtual Data &Back() = 0;
00143
00149     inline void Map(MapFun mapFun) override
00150     {
00151       PreOrderMap(mapFun); // Di default mappa in ordine PreOrder
00152     }
00153
00159     virtual void PreOrderMap(MapFun mapFun) override = 0;
00160
00166     virtual void PostOrderMap(MapFun mapFun) override = 0;
00167   };
00168
00176   template <typename Data>
00177   class SortableLinearContainer : virtual public LinearContainer<Data>
00178   {
00179   protected:
00180     using Container::size;
00181
00182   public:
00184     virtual ~SortableLinearContainer() noexcept = default;
00185
00186     // Copy assignment
00187     SortableLinearContainer &operator=(const SortableLinearContainer &) noexcept = delete;
00188
00189     // Move assignment
00190     SortableLinearContainer &operator=(SortableLinearContainer &&) noexcept = delete;
00191
00193     inline bool operator==(const SortableLinearContainer &) const noexcept;
00194
00196     inline bool operator!=(const SortableLinearContainer &) const noexcept;
00197
00201     inline void Sort() noexcept;
00202
00203   protected:
00210     void quickSort(ulong left, ulong right) noexcept;
00211
00219     ulong partition(ulong left, ulong right) noexcept;
00220   };
00221
00222 }
00223
00224 #include "linear.cpp"
00225
00226 #endif
```

7.35 Exercise1/zlasdtest/container/linear.hpp File Reference

```
#include "../../container/linear.hpp"
```

Functions

- template<typename Data>
  void EqualLinear (uint &testnum, uint &testerr, const lasd::LinearContainer< Data > &con1, const

lasd::LinearContainer< Data > &con2, bool chk)

- template<typename Data>
  void NonEqualLinear (uint &testnum, uint &testerr, const lasd::LinearContainer< Data > &con1, const lasd::LinearContainer< Data > &con2, bool chk)
- template<typename Data>
  void GetAt (uint &testnum, uint &testerr, const lasd::LinearContainer< Data > &con, bool chk, const ulong &ind, const Data &val)
- template<typename Data>
  void GetFront (uint &testnum, uint &testerr, const lasd::LinearContainer< Data > &con, bool chk, const Data &val)
- template<typename Data>
  void GetBack (uint &testnum, uint &testerr, const lasd::LinearContainer< Data > &con, bool chk, const Data &val)
- template<typename Data>
  void SetAt (uint &testnum, uint &testerr, lasd::MutableLinearContainer< Data > &con, bool chk, const ulong &ind, const Data &val)
- template<typename Data>
  void SetFront (uint &testnum, uint &testerr, lasd::MutableLinearContainer< Data > &con, bool chk, const Data &val)
- template<typename Data>
  void SetBack (uint &testnum, uint &testerr, lasd::MutableLinearContainer< Data > &con, bool chk, const Data &val)

7.35.1 Function Documentation

7.35.1.1 EqualLinear()

```
template<typename Data>
void EqualLinear (
             uint & testnum,
             uint & testerr,
             const lasd::LinearContainer< Data > & con1,
             const lasd::LinearContainer< Data > & con2,
             bool chk)
```

Definition at line 12 of file linear.hpp.

7.35.1.2 GetAt()

```
template<typename Data>
void GetAt (
             uint & testnum,
             uint & testerr,
             const lasd::LinearContainer< Data > & con,
             bool chk,
             const ulong & ind,
             const Data & val)
```

Definition at line 40 of file linear.hpp.

7.35.1.3 GetBack()

```
template<typename Data>
void GetBack (
             uint & testnum,
             uint & testerr,
             const lasd::LinearContainer< Data > & con,
             bool chk,
             const Data & val)
```

Definition at line 76 of file linear.hpp.

7.35.1.4 GetFront()

```
template<typename Data>
void GetFront (
             uint & testnum,
             uint & testerr,
             const lasd::LinearContainer< Data > & con,
             bool chk,
             const Data & val)
```

Definition at line 58 of file linear.hpp.

7.35.1.5 NonEqualLinear()

```
template<typename Data>
void NonEqualLinear (
             uint & testnum,
             uint & testerr,
             const lasd::LinearContainer< Data > & con1,
             const lasd::LinearContainer< Data > & con2,
             bool chk)
```

Definition at line 26 of file linear.hpp.

7.35.1.6 SetAt()

```
template<typename Data>
void SetAt (
             uint & testnum,
             uint & testerr,
             lasd::MutableLinearContainer< Data > & con,
             bool chk,
             const ulong & ind,
             const Data & val)
```

Definition at line 98 of file linear.hpp.

7.35.1.7 SetBack()

```
template<typename Data>
void SetBack (
              uint & testnum,
              uint & testerr,
              lasd::MutableLinearContainer< Data > & con,
              bool chk,
              const Data & val)
```

Definition at line 136 of file linear.hpp.

7.35.1.8 SetFront()

```
template<typename Data>
void SetFront (
              uint & testnum,
              uint & testerr,
              lasd::MutableLinearContainer< Data > & con,
              bool chk,
              const Data & val)
```

Definition at line 117 of file linear.hpp.

7.36 linear.hpp

Go to the documentation of this file.

```
00001
00002 #ifndef LINEARTEST_HPP
00003 #define LINEARTEST_HPP
00004
00005 #include "../../container/linear.hpp"
00006
00007 /* ********************************************************************** */
00008
00009 // LinearContainer member functions!
00010
00011 template <typename Data>
00012 void EqualLinear(uint & testnum, uint & testerr, const lasd::LinearContainer<Data> & con1, const
      lasd::LinearContainer<Data> & con2, bool chk) {
00013   bool tst;
00014   testnum++;
00015   try {
00016     std::cout « " " « testnum « " (" « testerr « ") The two linear containers are " « ((tst = (con1 ==
      con2)) ? "" : "not ") « "equal: ";
00017     std::cout « ((tst = (tst == chk)) ? "Correct" : "Error") « "!" « std::endl;
00018   }
00019   catch (std::exception & exc) {
00020     std::cout « "\"" « exc.what() « "\": " « ((tst = !chk) ? "Correct" : "Error") « "!" « std::endl;
00021   }
00022   testerr += (1 - (uint) tst);
00023 }
00024
00025 template <typename Data>
00026 void NonEqualLinear(uint & testnum, uint & testerr, const lasd::LinearContainer<Data> & con1, const
      lasd::LinearContainer<Data> & con2, bool chk) {
00027   bool tst;
00028   testnum++;
00029   try {
00030     std::cout « " " « testnum « " (" « testerr « ") The two linear containers are " « ((tst = (con1 !=
      con2)) ? "not " : "") « "equal: ";
00031     std::cout « ((tst = (tst == chk)) ? "Correct" : "Error") « "!" « std::endl;
00032   }
00033   catch (std::exception & exc) {
00034     std::cout « "\"" « exc.what() « "\": " « ((tst = !chk) ? "Correct" : "Error") « "!" « std::endl;
00035   }
```

```
00036   testerr += (1 - (uint) tst);
00037 }
00038
00039 template <typename Data>
00040 void GetAt(uint & testnum, uint & testerr, const lasd::LinearContainer<Data> & con, bool chk, const
      ulong & ind, const Data & val) {
00041   bool tst;
00042   testnum++;
00043   try {
00044     std::cout << " " << testnum << " (" << testerr << ") Get of the linear container at index \"" << ind <<
      "\" with value \"" << con[ind] << "\": ";
00045     std::cout << ((tst = ((con[ind] == val) == chk)) ? "Correct" : "Error") << "!" << std::endl;
00046   }
00047   catch (std::out_of_range & exc) {
00048     std::cout << exc.what() << "\": " << ((tst = !chk) ? "Correct" : "Error") << "!" << std::endl;
00049   }
00050   catch (std::exception & exc) {
00051     tst = false;
00052     std::cout << std::endl << "Wrong exception: " << exc.what() << "!" << std::endl;
00053   }
00054   testerr += (1 - (uint) tst);
00055 }
00056
00057 template <typename Data>
00058 void GetFront(uint & testnum, uint & testerr, const lasd::LinearContainer<Data> & con, bool chk, const
      Data & val) {
00059   bool tst;
00060   testnum++;
00061   try {
00062     std::cout << " " << testnum << " (" << testerr << ") The front of the linear container is \"" <<
      con.Front() << "\": ";
00063     std::cout << ((tst = ((con.Front() == val) == chk)) ? "Correct" : "Error") << "!" << std::endl;
00064   }
00065   catch (std::length_error & exc) {
00066     std::cout << exc.what() << "\": " << ((tst = !chk) ? "Correct" : "Error") << "!" << std::endl;
00067   }
00068   catch (std::exception & exc) {
00069     tst = false;
00070     std::cout << std::endl << "Wrong exception: " << exc.what() << "!" << std::endl;
00071   }
00072   testerr += (1 - (uint) tst);
00073 }
00074
00075 template <typename Data>
00076 void GetBack(uint & testnum, uint & testerr, const lasd::LinearContainer<Data> & con, bool chk, const
      Data & val) {
00077   bool tst;
00078   testnum++;
00079   try {
00080     std::cout << " " << testnum << " (" << testerr << ") The back of the linear container is \"" <<
      con.Back() << "\": ";
00081     std::cout << ((tst = ((con.Back() == val) == chk)) ? "Correct" : "Error") << "!" << std::endl;
00082   }
00083   catch (std::length_error & exc) {
00084     std::cout << exc.what() << "\": " << ((tst = !chk) ? "Correct" : "Error") << "!" << std::endl;
00085   }
00086   catch (std::exception & exc) {
00087     tst = false;
00088     std::cout << std::endl << "Wrong exception: " << exc.what() << "!" << std::endl;
00089   }
00090   testerr += (1 - (uint) tst);
00091 }
00092
00093 /* ********************************************************************** */
00094
00095 // MutableLinearContainer member functions!
00096
00097 template <typename Data>
00098 void SetAt(uint & testnum, uint & testerr, lasd::MutableLinearContainer<Data> & con, bool chk, const
      ulong & ind, const Data & val) {
00099   bool tst;
00100   testnum++;
00101   try {
00102     std::cout << " " << testnum << " (" << testerr << ") Set of the linear container at index \"" << ind <<
      "\" with value \"" << val << "\": ";
00103     con[ind] = val;
00104     std::cout << ((tst = ((con[ind] == val) == chk)) ? "Correct" : "Error") << "!" << std::endl;
00105   }
00106   catch (std::out_of_range & exc) {
00107     std::cout << "\"" << exc.what() << "\": " << ((tst = !chk) ? "Correct" : "Error") << "!" << std::endl;
00108   }
00109   catch (std::exception & exc) {
00110     tst = false;
00111     std::cout << std::endl << "Wrong exception: " << exc.what() << "!" << std::endl;
00112   }
00113   testerr += (1 - (uint) tst);
00114 }
```

```
00115
00116 template <typename Data>
00117 void SetFront(uint & testnum, uint & testerr, lasd::MutableLinearContainer<Data> & con, bool chk,
      const Data & val) {
00118   bool tst;
00119   testnum++;
00120   try {
00121     std::cout « " " « testnum « " (" « testerr « ") Setting the front of the linear container to \"" «
      val « "\": ";
00122     con.Front() = val;
00123     std::cout « ((tst = ((con.Front() == val) == chk)) ? "Correct" : "Error") « "!" « std::endl;
00124   }
00125   catch (std::length_error & exc) {
00126     std::cout « exc.what() « "\": " « ((tst = !chk) ? "Correct" : "Error") « "!" « std::endl;
00127   }
00128   catch (std::exception & exc) {
00129     tst = false;
00130     std::cout « std::endl « "Wrong exception: " « exc.what() « "!" « std::endl;
00131   }
00132   testerr += (1 - (uint) tst);
00133 }
00134
00135 template <typename Data>
00136 void SetBack(uint & testnum, uint & testerr, lasd::MutableLinearContainer<Data> & con, bool chk, const
      Data & val) {
00137   bool tst;
00138   testnum++;
00139   try {
00140     std::cout « " " « testnum « " (" « testerr « ") Setting the back of the linear container to \"" «
      val « "\": ";
00141     con.Back() = val;
00142     std::cout « ((tst = ((con.Back() == val) == chk)) ? "Correct" : "Error") « "!" « std::endl;
00143   }
00144   catch (std::length_error & exc) {
00145     std::cout « exc.what() « "\": " « ((tst = !chk) ? "Correct" : "Error") « "!" « std::endl;
00146   }
00147   catch (std::exception & exc) {
00148     tst = false;
00149     std::cout « std::endl « "Wrong exception: " « exc.what() « "!" « std::endl;
00150   }
00151   testerr += (1 - (uint) tst);
00152 }
00153
00154 /* ********************************************************************************** */
00155
00156 #endif
```

7.37 Exercise1/container/mappable.hpp File Reference

```
#include <functional>
#include "traversable.hpp"
#include "mappable.cpp"
```

Classes

- class lasd::MappableContainer< Data >

  Classe astratta che estende TraversableContainer, permettendo la modifica degli elementi tramite funzioni mappanti.
- class lasd::PreOrderMappableContainer< Data >

  Estensione di MappableContainer che specifica l'ordine PreOrder per la mappatura.
- class lasd::PostOrderMappableContainer< Data >

  Estensione di MappableContainer che specifica l'ordine PostOrder per la mappatura.

Namespaces

- namespace lasd

7.38 mappable.hpp

Go to the documentation of this file.

```
00001 #ifndef MAPPABLE_HPP
00002 #define MAPPABLE_HPP
00003
00004 #include <functional>
00005 #include "traversable.hpp"
00006
00007 namespace lasd
00008 {
00009
00015   template <typename Data>
00016   class MappableContainer : virtual public TraversableContainer<Data>
00017   {
00018
00019   public:
00021     virtual ~MappableContainer() noexcept = default;
00022
00023     // Copy assignment
00024     MappableContainer &operator=(const MappableContainer &) noexcept = delete;
00025
00026     // Move assignment
00027     MappableContainer &operator=(MappableContainer &&) noexcept = delete;
00028
00030     bool operator==(const MappableContainer &) const noexcept = delete;
00031
00033     bool operator!=(const MappableContainer &) const noexcept = delete;
00034
00038     using MapFun = std::function<void(Data &)>;
00039
00045     virtual void Map(const MapFun fun) = 0;
00046   };
00047
00053   template <typename Data>
00054   class PreOrderMappableContainer
00055       : virtual public MappableContainer<Data>,
00056         virtual public PreOrderTraversableContainer<Data>
00057   {
00058
00059   public:
00061     virtual ~PreOrderMappableContainer() = default;
00062
00063     // Copy assignment
00064     PreOrderMappableContainer &operator=(const PreOrderMappableContainer &) = delete;
00065
00066     // Move assignment
00067     PreOrderMappableContainer &operator=(PreOrderMappableContainer &&) noexcept = delete;
00068
00070     bool operator==(const PreOrderMappableContainer &) const noexcept = delete;
00071
00073     bool operator!=(const PreOrderMappableContainer &) const noexcept = delete;
00074
00080     virtual void PreOrderMap(MapFun fun) = 0;
00081
00087     inline void Map(MapFun fun) override;
00088   };
00089
00095   template <typename Data>
00096   class PostOrderMappableContainer
00097       : virtual public MappableContainer<Data>,
00098         virtual public PostOrderTraversableContainer<Data>
00099   {
00100
00101   public:
00103     virtual ~PostOrderMappableContainer() = default;
00104
00105     // Copy assignment
00106     PostOrderMappableContainer &operator=(const PostOrderMappableContainer &) = delete;
00107
00108     // Move assignment
00109     PostOrderMappableContainer &operator=(PostOrderMappableContainer &&) noexcept = delete;
00110
00112     bool operator==(const PostOrderMappableContainer &) const noexcept = delete;
00113
00115     bool operator!=(const PostOrderMappableContainer &) const noexcept = delete;
00116
00122     virtual void PostOrderMap(const MapFun fun) = 0;
00123
00129     inline void Map(const MapFun fun) override;
00130   };
00131
00132 }
00133
00134 #include "mappable.cpp"
```

```
00135
00136 #endif
```

7.39 Exercise1/zlasdtest/container/mappable.hpp File Reference

```
#include "../../container/mappable.hpp"
```

Functions

- template<typename Data>
  void Map (uint &testnum, uint &testerr, lasd::MappableContainer< Data > &con, bool chk, typename lasd::MappableContainer< Data >::MapFun fun)
- template<typename Data>
  void MapIncrement (Data &dat)
- template<typename Data>
  void MapDecrement (Data &dat)
- template<typename Data>
  void MapIncrementNPrint (Data &dat)
- template<typename Data>
  void MapDouble (Data &dat)
- template<typename Data>
  void MapHalf (Data &dat)
- template<typename Data>
  void MapDoubleNPrint (Data &dat)
- template<typename Data>
  void MapInvert (Data &dat)
- template<typename Data>
  void MapInvertNPrint (Data &dat)
- template<typename Data>
  void MapParityInvert (Data &dat)
- void MapStringAppend (std::string &, const std::string &)
- void MapStringNonEmptyAppend (std::string &, const std::string &)
- template<typename Data>
  void MapPreOrder (uint &testnum, uint &testerr, lasd::PreOrderMappableContainer< Data > &con, bool chk, typename lasd::MappableContainer< Data >::MapFun fun)
- template<typename Data>
  void MapPostOrder (uint &testnum, uint &testerr, lasd::PostOrderMappableContainer< Data > &con, bool chk, typename lasd::MappableContainer< Data >::MapFun fun)

7.39.1 Function Documentation

7.39.1.1 Map()

```
template<typename Data>
void Map (
            uint & testnum,
            uint & testerr,
            lasd::MappableContainer< Data > & con,
            bool chk,
            typename lasd::MappableContainer< Data >::MapFun fun)
```

Definition at line 12 of file mappable.hpp.

7.39.1.2 MapDecrement()

```
template<typename Data>
void MapDecrement (
             Data & dat)
```

Definition at line 32 of file mappable.hpp.

7.39.1.3 MapDouble()

```
template<typename Data>
void MapDouble (
             Data & dat)
```

Definition at line 42 of file mappable.hpp.

7.39.1.4 MapDoubleNPrint()

```
template<typename Data>
void MapDoubleNPrint (
             Data & dat)
```

Definition at line 52 of file mappable.hpp.

7.39.1.5 MapHalf()

```
template<typename Data>
void MapHalf (
             Data & dat)
```

Definition at line 47 of file mappable.hpp.

7.39.1.6 MapIncrement()

```
template<typename Data>
void MapIncrement (
             Data & dat)
```

Definition at line 27 of file mappable.hpp.

7.39.1.7 MapIncrementNPrint()

```
template<typename Data>
void MapIncrementNPrint (
             Data & dat)
```

Definition at line 37 of file mappable.hpp.

7.39.1.8 MapInvert()

```
template<typename Data>
void MapInvert (
             Data & dat)
```

Definition at line 57 of file mappable.hpp.

7.39.1.9 MapInvertNPrint()

```
template<typename Data>
void MapInvertNPrint (
             Data & dat)
```

Definition at line 62 of file mappable.hpp.

7.39.1.10 MapParityInvert()

```
template<typename Data>
void MapParityInvert (
             Data & dat)
```

Definition at line 67 of file mappable.hpp.

7.39.1.11 MapPostOrder()

```
template<typename Data>
void MapPostOrder (
             uint & testnum,
             uint & testerr,
             lasd::PostOrderMappableContainer< Data > & con,
             bool chk,
             typename lasd::MappableContainer< Data >::MapFun fun)
```

Definition at line 99 of file mappable.hpp.

7.39.1.12 MapPreOrder()

```
template<typename Data>
void MapPreOrder (
             uint & testnum,
             uint & testerr,
             lasd::PreOrderMappableContainer< Data > & con,
             bool chk,
             typename lasd::MappableContainer< Data >::MapFun fun)
```

Definition at line 80 of file mappable.hpp.

7.39.1.13 MapStringAppend()

```
void MapStringAppend (
            std::string & dat,
            const std::string & par)
```

Definition at line 46 of file container.cpp.

7.39.1.14 MapStringNonEmptyAppend()

```
void MapStringNonEmptyAppend (
            std::string & dat,
            const std::string & par)
```

Definition at line 50 of file container.cpp.

7.40 mappable.hpp

Go to the documentation of this file.

```
00001
00002 #ifndef MAPPABLETEST_HPP
00003 #define MAPPABLETEST_HPP
00004
00005 #include "../../container/mappable.hpp"
00006
00007 /* ************************************************************************** */
00008
00009 // MappableContainer member functions!
00010
00011 template <typename Data>
00012 void Map(uint & testnum, uint & testerr, lasd::MappableContainer<Data> & con, bool chk, typename
      lasd::MappableContainer<Data>::MapFun fun) {
00013   bool tst = true;
00014   testnum++;
00015   try {
00016     std::cout << " " << testnum << " (" << testerr << ") Executing map - ";
00017     con.Map(fun);
00018     std::cout << ": " << ((tst = chk) ? "Correct" : "Error") << "!" << std::endl;
00019   }
00020   catch (std::exception & exc) {
00021     std::cout << "\"" << exc.what() << "\": " << ((tst = !chk) ? "Correct" : "Error") << "!" << std::endl;
00022   }
00023   testerr += (1 - (uint) tst);
00024 }
00025
00026 template <typename Data>
00027 void MapIncrement(Data & dat) {
00028   dat++;
00029 }
00030
00031 template <typename Data>
00032 void MapDecrement(Data & dat) {
00033   dat--;
00034 }
00035
00036 template <typename Data>
00037 void MapIncrementNPrint(Data & dat) {
00038   std::cout << dat++ << "->" << dat << "; ";
00039 }
00040
00041 template <typename Data>
00042 void MapDouble(Data & dat) {
00043   dat *= 2;
00044 }
00045
00046 template <typename Data>
00047 void MapHalf(Data & dat) {
00048   dat /= 2;
00049 }
00050
```

```
00051 template <typename Data>
00052 void MapDoubleNPrint(Data & dat) {
00053   std::cout « dat « "->" « (dat *= 2) « "; ";
00054 }
00055
00056 template <typename Data>
00057 void MapInvert(Data & dat) {
00058   dat = -dat;
00059 }
00060
00061 template <typename Data>
00062 void MapInvertNPrint(Data & dat) {
00063   std::cout « dat « "->" « (dat = -dat) « "; ";
00064 }
00065
00066 template <typename Data>
00067 void MapParityInvert(Data & dat) {
00068   if (dat % 2 != 0) { dat = -dat; }
00069 }
00070
00071 void MapStringAppend(std::string &, const std::string &);
00072
00073 void MapStringNonEmptyAppend(std::string &, const std::string &);
00074
00075 /* ********************************************************************** */
00076
00077 // PreOrderMappableContainer member functions!
00078
00079 template <typename Data>
00080 void MapPreOrder(uint & testnum, uint & testerr, lasd::PreOrderMappableContainer<Data> & con, bool
      chk, typename lasd::MappableContainer<Data>::MapFun fun) {
00081   bool tst = true;
00082   testnum++;
00083   try {
00084     std::cout « " " « testnum « " (" « testerr « ") Executing map in pre order - ";
00085     con.PreOrderMap(fun);
00086     std::cout « ": " « ((tst = chk) ? "Correct" : "Error") « "!" « std::endl;
00087   }
00088   catch (std::exception & exc) {
00089     std::cout « "\"" « exc.what() « "\": " « ((tst = !chk) ? "Correct" : "Error") « "!" « std::endl;
00090   }
00091   testerr += (1 - (uint) tst);
00092 }
00093
00094 /* ********************************************************************** */
00095
00096 // PostOrderMappableContainer member functions!
00097
00098 template <typename Data>
00099 void MapPostOrder(uint & testnum, uint & testerr, lasd::PostOrderMappableContainer<Data> & con, bool
      chk, typename lasd::MappableContainer<Data>::MapFun fun) {
00100   bool tst = true;
00101   testnum++;
00102   try {
00103     std::cout « " " « testnum « " (" « testerr « ") Executing map in post order - ";
00104     con.PostOrderMap(fun);
00105     std::cout « ": " « ((tst = chk) ? "Correct" : "Error") « "!" « std::endl;
00106   }
00107   catch (std::exception & exc) {
00108     std::cout « "\"" « exc.what() « "\": " « ((tst = !chk) ? "Correct" : "Error") « "!" « std::endl;
00109   }
00110   testerr += (1 - (uint) tst);
00111 }
00112
00113 /* ********************************************************************** */
00114
00115 #endif
```

7.41 Exercise1/container/testable.hpp File Reference

```
#include "container.hpp"
```

Classes

- class lasd::TestableContainer< Data >

  Classe astratta che estende Container con un metodo per il test di esistenza.

Namespaces

- namespace lasd

7.42 testable.hpp

Go to the documentation of this file.

```
00001 #ifndef TESTABLE_HPP
00002 #define TESTABLE_HPP
00003 #include "container.hpp"
00004
00005 namespace lasd
00006 {
00007
00010   template <typename Data>
00011   class TestableContainer : virtual public Container
00012   {
00013
00014   private:
00015     // Dati membri riservati (se presenti)
00016
00017   protected:
00019     TestableContainer() = default;
00020
00021   public:
00023     virtual ~TestableContainer() = default;
00024
00025     // Copy assignment
00026     TestableContainer &operator=(const TestableContainer &) = delete;
00027
00028     // Move assignment
00029     TestableContainer &operator=(TestableContainer &&) noexcept = delete;
00030
00031     // Comparison operators
00032
00034     bool operator==(const TestableContainer &) const noexcept = delete;
00035
00037     bool operator!=(const TestableContainer &) const noexcept = delete;
00038
00039
00046     virtual bool Exists(const Data &dato) const noexcept = 0;
00047   };
00048
00049
00050 }
00051
00052 #endif
```

7.43 Exercise1/zlasdtest/container/testable.hpp File Reference

```
#include "../../container/testable.hpp"
```

Functions

- template<typename Data>
  void Exists (uint &testnum, uint &testerr, const lasd::TestableContainer< Data > &con, bool chk, const Data &val)

7.43.1 Function Documentation

7.43.1.1 Exists()

```
template<typename Data>
void Exists (
             uint & testnum,
             uint & testerr,
             const lasd::TestableContainer< Data > & con,
             bool chk,
             const Data & val)
```

Definition at line 12 of file testable.hpp.

7.44 testable.hpp

Go to the documentation of this file.

```
00001
00002 #ifndef TESTABLETEST_HPP
00003 #define TESTABLETEST_HPP
00004
00005 #include "../../container/testable.hpp"
00006
00007 /* ********************************************************************** */
00008
00009 // TestableContainer member functions!
00010
00011 template <typename Data>
00012 void Exists(uint & testnum, uint & testerr, const lasd::TestableContainer<Data> & con, bool chk, const
      Data & val) {
00013   bool tst;
00014   testnum++;
00015   std::cout « " " « testnum « " (" « testerr « ") Data \"" « val « "\" " « ((tst = con.Exists(val)) ?
      "does" : "does not") « " exist: ";
00016   std::cout « ((tst = (tst == chk)) ? "Correct" : "Error") « "!" « std::endl;
00017   testerr += (1 - (uint) tst);
00018 }
00019
00020 /* ********************************************************************** */
00021
00022 #endif
```

7.45 Exercise1/container/traversable.hpp File Reference

```
#include <functional>
#include "testable.hpp"
```

Classes

- class lasd::TraversableContainer< Data >

  Classe astratta di contenitore traversabile.
- class lasd::PreOrderTraversableContainer< Data >

  Classe astratta per contenitori con traversata in pre-ordine.
- class lasd::PostOrderTraversableContainer< Data >

  Classe astratta per contenitori con traversata in post-ordine.

Namespaces

- namespace lasd

7.46 traversable.hpp

Go to the documentation of this file.

```
00001 #ifndef TRAVERSABLE_HPP
00002 #define TRAVERSABLE_HPP
00003
00004 #include <functional>
00005 #include "testable.hpp"
00006 namespace lasd
00007 {
00010   template <typename Data>
00011   class TraversableContainer : virtual public TestableContainer<Data>
00012   {
00013
00014   public:
00016     virtual ~TraversableContainer() = default;
00017
00018     // Copy assignment
00019     TraversableContainer &operator=(const TraversableContainer &) = delete;
00020
00021     // Move assignment
00022     TraversableContainer &operator=(TraversableContainer &&) noexcept = delete;
00023
00024     // Comparison operators
00025     bool operator==(const TraversableContainer &) const noexcept = delete;
00026     bool operator!=(const TraversableContainer &) const noexcept = delete;
00027
00029     using TraverseFun = std::function<void(const Data &)>;
00030
00033     virtual void Traverse(TraverseFun func) const = 0;
00034
00036     template <typename Accumulator>
00037     using FoldFun = std::function<Accumulator(const Data &, const Accumulator &)>;
00038
00044     template <typename Accumulator>
00045     inline Accumulator Fold(FoldFun<Accumulator> func, Accumulator base) const;
00046
00050     inline bool Exists(const Data &elem) const noexcept override;
00051   };
00052
00055   template <typename Data>
00056   class PreOrderTraversableContainer : virtual public TraversableContainer<Data>
00057   {
00058
00059   public:
00061     virtual ~PreOrderTraversableContainer() = default;
00062
00063     // Copy assignment
00064     PreOrderTraversableContainer &operator=(const PreOrderTraversableContainer &) = delete;
00065
00066     // Move assignment
00067     PreOrderTraversableContainer &operator=(PreOrderTraversableContainer &&) noexcept = delete;
00068
00069     // Comparison operators
00070     bool operator==(const PreOrderTraversableContainer &) const noexcept = delete;
00071     bool operator!=(const PreOrderTraversableContainer &) const noexcept = delete;
00072
00073
00076     virtual void PreOrderTraverse(const TraverseFun func) const = 0;
00077
00079     template <typename Accumulator>
00080     using FoldFun = typename TraversableContainer<Data>::FoldFun<Accumulator>;
00081
00087     template <typename Accumulator>
00088     inline Accumulator PreOrderFold(FoldFun<Accumulator> func, Accumulator base) const;
00089
00092     inline void Traverse(TraverseFun func) const override;
00093   };
00094
00097   template <typename Data>
00098   class PostOrderTraversableContainer : virtual public TraversableContainer<Data>
00099   {
00100
00101   public:
00103     virtual ~PostOrderTraversableContainer() = default;
00104
```

```
00105     // Copy assignment
00106     PostOrderTraversableContainer &operator=(const PostOrderTraversableContainer &) = delete;
00107
00108     // Move assignment
00109     PostOrderTraversableContainer &operator=(PostOrderTraversableContainer &&) noexcept = delete;
00110
00111     // Comparison operators
00112     bool operator==(const PostOrderTraversableContainer &) const noexcept = delete;
00113     bool operator!=(const PostOrderTraversableContainer &) const noexcept = delete;
00114
00117     virtual void PostOrderTraverse(TraverseFun func) const = 0;
00118
00120     template <typename Accumulator>
00121     using FoldFun = typename TraversableContainer<Data>::FoldFun<Accumulator>;
00122
00128     template <typename Accumulator>
00129     inline Accumulator PostOrderFold(FoldFun<Accumulator> func, Accumulator base) const;
00130
00133     inline void Traverse(TraverseFun func) const override;
00134   };
00135
00136 }
00137
00138 #endif
```

7.47 Exercise1/zlasdtest/container/traversable.hpp File Reference

```
#include "../../container/traversable.hpp"
```

Functions

- template<typename Data>
  void Traverse (uint &testnum, uint &testerr, const lasd::TraversableContainer< Data > &con, bool chk, typename lasd::TraversableContainer< Data >::TraverseFun fun)
- template<typename Data, typename Value>
  void Fold (uint &testnum, uint &testerr, const lasd::TraversableContainer< Data > &con, bool chk, typename lasd::TraversableContainer< Data >::FoldFun< Value > fun, const Value &inival, const Value &finval)
- template<typename Data>
  void TraversePrint (const Data &dat)
- template<typename Data>
  Data FoldAdd (const Data &dat, const Data &acc)
- template<typename Data>
  Data FoldMultiply (const Data &dat, const Data &acc)
- int FoldParity (const int &, const int &)
- std::string FoldStringConcatenate (const std::string &, const std::string &)
- template<typename Data>
  void TraversePreOrder (uint &testnum, uint &testerr, const lasd::PreOrderTraversableContainer< Data > &con, bool chk, typename lasd::TraversableContainer< Data >::TraverseFun fun)
- template<typename Data, typename Value>
  void FoldPreOrder (uint &testnum, uint &testerr, const lasd::PreOrderTraversableContainer< Data > &con, bool chk, typename lasd::TraversableContainer< Data >::FoldFun< Value > fun, const Value &inival, const Value &finval)
- template<typename Data>
  void TraversePostOrder (uint &testnum, uint &testerr, const lasd::PostOrderTraversableContainer< Data > &con, bool chk, typename lasd::TraversableContainer< Data >::TraverseFun fun)
- template<typename Data, typename Value>
  void FoldPostOrder (uint &testnum, uint &testerr, const lasd::PostOrderTraversableContainer< Data > &con, bool chk, typename lasd::TraversableContainer< Data >::FoldFun< Value > fun, const Value &inival, const Value &finval)

7.47.1 Function Documentation

7.47.1.1 Fold()

```
template<typename Data, typename Value>
void Fold (
            uint & testnum,
            uint & testerr,
            const lasd::TraversableContainer< Data > & con,
            bool chk,
            typename lasd::TraversableContainer< Data >::FoldFun< Value > fun,
            const Value & inival,
            const Value & finval)
```

Definition at line 27 of file traversable.hpp.

7.47.1.2 FoldAdd()

```
template<typename Data>
Data FoldAdd (
            const Data & dat,
            const Data & acc)
```

Definition at line 48 of file traversable.hpp.

7.47.1.3 FoldMultiply()

```
template<typename Data>
Data FoldMultiply (
            const Data & dat,
            const Data & acc)
```

Definition at line 53 of file traversable.hpp.

7.47.1.4 FoldParity()

```
int FoldParity (
            const int & dat,
            const int & acc)
```

Definition at line 32 of file container.cpp.

7.47.1.5 FoldPostOrder()

```
template<typename Data, typename Value>
void FoldPostOrder (
            uint & testnum,
            uint & testerr,
            const lasd::PostOrderTraversableContainer< Data > & con,
            bool chk,
            typename lasd::TraversableContainer< Data >::FoldFun< Value > fun,
            const Value & inival,
            const Value & finval)
```

Definition at line 116 of file traversable.hpp.

7.47.1.6 FoldPreOrder()

```
template<typename Data, typename Value>
void FoldPreOrder (
             uint & testnum,
             uint & testerr,
             const lasd::PreOrderTraversableContainer< Data > & con,
             bool chk,
             typename lasd::TraversableContainer< Data >::FoldFun< Value > fun,
             const Value & inival,
             const Value & finval)
```

Definition at line 81 of file traversable.hpp.

7.47.1.7 FoldStringConcatenate()

```
std::string FoldStringConcatenate (
             const std::string & dat,
             const std::string & acc)
```

Definition at line 36 of file container.cpp.

7.47.1.8 Traverse()

```
template<typename Data>
void Traverse (
             uint & testnum,
             uint & testerr,
             const lasd::TraversableContainer< Data > & con,
             bool chk,
             typename lasd::TraversableContainer< Data >::TraverseFun fun)
```

Definition at line 12 of file traversable.hpp.

7.47.1.9 TraversePostOrder()

```
template<typename Data>
void TraversePostOrder (
             uint & testnum,
             uint & testerr,
             const lasd::PostOrderTraversableContainer< Data > & con,
             bool chk,
             typename lasd::TraversableContainer< Data >::TraverseFun fun)
```

Definition at line 101 of file traversable.hpp.

7.47.1.10 TraversePreOrder()

```
template<typename Data>
void TraversePreOrder (
             uint & testnum,
             uint & testerr,
             const lasd::PreOrderTraversableContainer< Data > & con,
             bool chk,
             typename lasd::TraversableContainer< Data >::TraverseFun fun)
```

Definition at line 66 of file traversable.hpp.

7.47.1.11 TraversePrint()

```
template<typename Data>
void TraversePrint (
            const Data & dat)
```

Definition at line 43 of file traversable.hpp.

7.48 traversable.hpp

Go to the documentation of this file.

```
00001
00002 #ifndef TRAVERSABLETEST_HPP
00003 #define TRAVERSABLETEST_HPP
00004
00005 #include "../../container/traversable.hpp"
00006
00007 /* ************************************************************************** */
00008
00009 // TraversableContainer member functions!
00010
00011 template <typename Data>
00012 void Traverse(uint & testnum, uint & testerr, const lasd::TraversableContainer<Data> & con, bool chk,
      typename lasd::TraversableContainer<Data>::TraverseFun fun) {
00013   bool tst = true;
00014   testnum++;
00015   try {
00016     std::cout « " " « testnum « " (" « testerr « ") Executing traverse - ";
00017     con.Traverse(fun);
00018     std::cout « ": " « ((tst = chk) ? "Correct" : "Error") « "!" « std::endl;
00019   }
00020   catch (std::exception & exc) {
00021     std::cout « "\"" « exc.what() « "\": " « ((tst = !chk) ? "Correct" : "Error") « "!" « std::endl;
00022   }
00023   testerr += (1 - (uint) tst);
00024 }
00025
00026 template <typename Data, typename Value>
00027 void Fold(uint & testnum, uint & testerr, const lasd::TraversableContainer<Data> & con, bool chk,
      typename lasd::TraversableContainer<Data>::FoldFun<Value> fun, const Value & inival, const Value &
      finval) {
00028   bool tst;
00029   testnum++;
00030   try {
00031     std::cout « " " « testnum « " (" « testerr « ") Executing fold - ";
00032     Value val = con.Fold(fun, inival);
00033     std::cout « "obtained value is \"" « val « "\": ";
00034     std::cout « ((tst = ((val == finval) == chk)) ? "Correct" : "Error") « "!" « std::endl;
00035   }
00036   catch (std::exception & exc) {
00037     std::cout « "\"" « exc.what() « "\": " « ((tst = !chk) ? "Correct" : "Error") « "!" « std::endl;
00038   }
00039   testerr += (1 - (uint) tst);
00040 }
00041
00042 template <typename Data>
00043 void TraversePrint(const Data & dat) {
00044   std::cout « dat « " ";
00045 }
00046
00047 template <typename Data>
00048 Data FoldAdd(const Data & dat, const Data & acc) {
00049   return (acc + dat);
00050 }
00051
00052 template <typename Data>
00053 Data FoldMultiply(const Data & dat, const Data & acc) {
00054   return (acc * dat);
00055 }
00056
00057 int FoldParity(const int &, const int &);
00058
00059 std::string FoldStringConcatenate(const std::string &, const std::string &);
00060
00061 /* ************************************************************************** */
00062
00063 // PreOrderTraversableContainer member functions!
```

```
00064
00065 template <typename Data>
00066 void TraversePreOrder(uint & testnum, uint & testerr, const lasd::PreOrderTraversableContainer<Data> &
      con, bool chk, typename lasd::TraversableContainer<Data>::TraverseFun fun) {
00067   bool tst = true;
00068   testnum++;
00069   try {
00070     std::cout « " " « testnum « " (" « testerr « ") Executing traverse in pre order - ";
00071     con.PreOrderTraverse(fun);
00072     std::cout « ": " « ((tst = chk) ? "Correct" : "Error") « "!" « std::endl;
00073   }
00074   catch (std::exception & exc) {
00075     std::cout « "\"" « exc.what() « "\": " « ((tst = !chk) ? "Correct" : "Error") « "!" « std::endl;
00076   }
00077   testerr += (1 - (uint) tst);
00078 }
00079
00080 template <typename Data, typename Value>
00081 void FoldPreOrder(uint & testnum, uint & testerr, const lasd::PreOrderTraversableContainer<Data> &
      con, bool chk, typename lasd::TraversableContainer<Data>::FoldFun<Value> fun, const Value & inival,
      const Value & finval) {
00082   bool tst;
00083   testnum++;
00084   try {
00085     std::cout « " " « testnum « " (" « testerr « ") Executing fold in pre order - ";
00086     Value val = con.PreOrderFold(fun, inival);
00087     std::cout « "obtained value is \"" « val « "\": ";
00088     std::cout « ((tst = ((val == finval) == chk)) ? "Correct" : "Error") « "!" « std::endl;
00089   }
00090   catch (std::exception & exc) {
00091     std::cout « "\"" « exc.what() « "\": " « ((tst = !chk) ? "Correct" : "Error") « "!" « std::endl;
00092   }
00093   testerr += (1 - (uint) tst);
00094 }
00095
00096 /* ********************************************************************** */
00097
00098 // PostOrderTraversableContainer member functions!
00099
00100 template <typename Data>
00101 void TraversePostOrder(uint & testnum, uint & testerr, const lasd::PostOrderTraversableContainer<Data>
      & con, bool chk, typename lasd::TraversableContainer<Data>::TraverseFun fun) {
00102   bool tst = true;
00103   testnum++;
00104   try {
00105     std::cout « " " « testnum « " (" « testerr « ") Executing traverse in post order - ";
00106     con.PostOrderTraverse(fun);
00107     std::cout « ": " « ((tst = chk) ? "Correct" : "Error") « "!" « std::endl;
00108   }
00109   catch (std::exception & exc) {
00110     std::cout « "\"" « exc.what() « "\": " « ((tst = !chk) ? "Correct" : "Error") « "!" « std::endl;
00111   }
00112   testerr += (1 - (uint) tst);
00113 }
00114
00115 template <typename Data, typename Value>
00116 void FoldPostOrder(uint & testnum, uint & testerr, const lasd::PostOrderTraversableContainer<Data> &
      con, bool chk, typename lasd::TraversableContainer<Data>::FoldFun<Value> fun, const Value & inival,
      const Value & finval) {
00117   bool tst;
00118   testnum++;
00119   try {
00120     std::cout « " " « testnum « " (" « testerr « ") Executing fold in post order - ";
00121     Value val = con.PostOrderFold(fun, inival);
00122     std::cout « "obtained value is \"" « val « "\": ";
00123     std::cout « ((tst = ((val == finval) == chk)) ? "Correct" : "Error") « "!" « std::endl;
00124   }
00125   catch (std::exception & exc) {
00126     std::cout « "\"" « exc.what() « "\": " « ((tst = !chk) ? "Correct" : "Error") « "!" « std::endl;
00127   }
00128   testerr += (1 - (uint) tst);
00129 }
00130
00131 /* ********************************************************************** */
00132
00133 #endif
```

7.49 Exercise1/zlasdtest/exercise1a/fulltest.cpp File Reference

```
#include <iostream>
```

Functions

- void testFullExercise1A (uint &, uint &)

7.49.1 Function Documentation

7.49.1.1 testFullExercise1A()

```
void testFullExercise1A (
            uint & ,
            uint & )
```

Definition at line 6 of file fulltest.cpp.

7.50 fulltest.cpp

Go to the documentation of this file.

```
00001
00002 #include <iostream>
00003
00004 /* ********************************************************************** */
00005
00006 void testFullExercise1A(uint &, uint &) {
00007 }
```

7.51 Exercise1/zlasdtest/exercise1b/fulltest.cpp File Reference

```
#include <iostream>
```

Functions

- void testFullExercise1B (uint &, uint &)

7.51.1 Function Documentation

7.51.1.1 testFullExercise1B()

```
void testFullExercise1B (
            uint & ,
            uint & )
```

Definition at line 6 of file fulltest.cpp.

7.52 fulltest.cpp

Go to the documentation of this file.

```
00001
00002 #include <iostream>
00003
00004 /* ********************************************************************** */
00005
00006 void testFullExercise1B(uint &, uint &) {
00007 }
```

7.53 Exercise1/zlasdtest/exercise1a/simpletest.cpp File Reference

```
#include <iostream>
#include "../container/container.hpp"
#include "../container/testable.hpp"
#include "../container/traversable.hpp"
#include "../container/mappable.hpp"
#include "../container/linear.hpp"
#include "../vector/vector.hpp"
#include "../list/list.hpp"
```

Functions

- void stestVectorInt (uint &testnum, uint &testerr)
- void stestVectorDouble (uint &testnum, uint &testerr)
- void stestVectorString (uint &testnum, uint &testerr)
- void stestVector (uint &testnum, uint &testerr)
- void stestListInt (uint &testnum, uint &testerr)
- void stestListDouble (uint &testnum, uint &testerr)
- void stestListString (uint &testnum, uint &testerr)
- void stestList (uint &testnum, uint &testerr)
- void stestVectorListInt (uint &testnum, uint &testerr)
- void stestVectorListDouble (uint &testnum, uint &testerr)
- void stestVectorListString (uint &testnum, uint &testerr)
- void stestVectorList (uint &testnum, uint &testerr)
- void testSimpleExercise1A (uint &testnum, uint &testerr)

7.53.1 Function Documentation

7.53.1.1 stestList()

```
void stestList (
            uint & testnum,
            uint & testerr)
```

Definition at line 345 of file simpletest.cpp.

7.53.1.2 stestListDouble()

```
void stestListDouble (
            uint & testnum,
            uint & testerr)
```

Definition at line 262 of file simpletest.cpp.

7.53.1.3 stestListInt()

```
void stestListInt (
            uint & testnum,
            uint & testerr)
```

Definition at line 176 of file simpletest.cpp.

7.53.1.4 stestListString()

```
void stestListString (
            uint & testnum,
            uint & testerr)
```

Definition at line 299 of file simpletest.cpp.

7.53.1.5 stestVector()

```
void stestVector (
            uint & testnum,
            uint & testerr)
```

Definition at line 164 of file simpletest.cpp.

7.53.1.6 stestVectorDouble()

```
void stestVectorDouble (
            uint & testnum,
            uint & testerr)
```

Definition at line 83 of file simpletest.cpp.

7.53.1.7 stestVectorInt()

```
void stestVectorInt (
            uint & testnum,
            uint & testerr)
```

Definition at line 22 of file simpletest.cpp.

7.53.1.8 stestVectorList()

```
void stestVectorList (
             uint & testnum,
             uint & testerr)
```

Definition at line 486 of file simpletest.cpp.

7.53.1.9 stestVectorListDouble()

```
void stestVectorListDouble (
             uint & testnum,
             uint & testerr)
```

Definition at line 392 of file simpletest.cpp.

7.53.1.10 stestVectorListInt()

```
void stestVectorListInt (
             uint & testnum,
             uint & testerr)
```

Definition at line 357 of file simpletest.cpp.

7.53.1.11 stestVectorListString()

```
void stestVectorListString (
             uint & testnum,
             uint & testerr)
```

Definition at line 427 of file simpletest.cpp.

7.53.1.12 stestVectorString()

```
void stestVectorString (
             uint & testnum,
             uint & testerr)
```

Definition at line 112 of file simpletest.cpp.

7.53.1.13 testSimpleExercise1A()

```
void testSimpleExercise1A (
             uint & testnum,
             uint & testerr)
```

Definition at line 498 of file simpletest.cpp.

7.54 simpletest.cpp

Go to the documentation of this file.

```
00001
00002 #include <iostream>
00003
00004 /* ************************************************************************** */
00005
00006 #include "../container/container.hpp"
00007 #include "../container/testable.hpp"
00008 #include "../container/traversable.hpp"
00009 #include "../container/mappable.hpp"
00010 #include "../container/linear.hpp"
00011
00012 #include "../vector/vector.hpp"
00013
00014 #include "../list/list.hpp"
00015
00016 /* ************************************************************************** */
00017
00018 using namespace std;
00019
00020 /* ************************************************************************** */
00021
00022 void stestVectorInt(uint & testnum, uint & testerr) {
00023   uint loctestnum = 0, loctesterr = 0;
00024   cout « endl « "Begin of Vector<int> Test:" « endl;
00025   try {
00026     {
00027       lasd::SortableVector<int> vec;
00028       Empty(loctestnum, loctesterr, vec, true);
00029
00030       GetFront(loctestnum, loctesterr, vec, false, 0);
00031       GetBack(loctestnum, loctesterr, vec, false, 0);
00032       SetAt(loctestnum, loctesterr, vec, false, 1, 0);
00033       GetAt(loctestnum, loctesterr, vec, false, 2, 0);
00034
00035       Exists(loctestnum, loctesterr, vec, false, 0);
00036
00037       TraversePreOrder(loctestnum, loctesterr, vec, true, &TraversePrint<int>);
00038       TraversePostOrder(loctestnum, loctesterr, vec, true, &TraversePrint<int>);
00039
00040       FoldPreOrder(loctestnum, loctesterr, vec, true, &FoldAdd<int>, 0, 0);
00041       FoldPostOrder(loctestnum, loctesterr, vec, true, &FoldAdd<int>, 0, 0);
00042     }
00043     {
00044       lasd::SortableVector<int> vec(3);
00045       Empty(loctestnum, loctesterr, vec, false);
00046       Size(loctestnum, loctesterr, vec, true, 3);
00047
00048       SetAt(loctestnum, loctesterr, vec, true, 0, 4);
00049       SetAt(loctestnum, loctesterr, vec, true, 1, 3);
00050       SetAt(loctestnum, loctesterr, vec, true, 2, 1);
00051
00052       GetFront(loctestnum, loctesterr, vec, true, 4);
00053       GetBack(loctestnum, loctesterr, vec, true, 1);
00054
00055       SetFront(loctestnum, loctesterr, vec, true, 5);
00056       SetBack(loctestnum, loctesterr, vec, true, 4);
00057
00058       Exists(loctestnum, loctesterr, vec, true, 4);
00059
00060       TraversePreOrder(loctestnum, loctesterr, vec, true, &TraversePrint<int>);
00061       TraversePostOrder(loctestnum, loctesterr, vec, true, &TraversePrint<int>);
00062       FoldPreOrder(loctestnum, loctesterr, vec, true, &FoldAdd<int>, 0, 12);
00063       FoldPostOrder(loctestnum, loctesterr, vec, true, &FoldMultiply<int>, 1, 60);
00064
00065       vec.Sort();
00066
00067       TraversePreOrder(loctestnum, loctesterr, vec, true, &TraversePrint<int>);
00068       TraversePostOrder(loctestnum, loctesterr, vec, true, &TraversePrint<int>);
00069
00070       vec.Resize(2);
00071       FoldPostOrder(loctestnum, loctesterr, vec, true, &FoldMultiply<int>, 1, 12);
00072     }
00073   }
00074   catch (...) {
00075     loctestnum++; loctesterr++;
00076     cout « endl « "Unmanaged error! " « endl;
00077   }
00078   cout « "End of Vector<int> Test! (Errors/Tests: " « loctesterr « "/" « loctestnum « ")" « endl;
00079   testnum += loctestnum;
00080   testerr += loctesterr;
00081 }
00082
```

```
00083 void stestVectorDouble(uint & testnum, uint & testerr) {
00084   uint loctestnum = 0, loctesterr = 0;
00085   cout « endl « "Begin of Vector<double> Test:" « endl;
00086   try {
00087     lasd::SortableVector<double> vec(3);
00088     Empty(loctestnum, loctesterr, vec, false);
00089     Size(loctestnum, loctesterr, vec, true, 3);
00090
00091     SetAt(loctestnum, loctesterr, vec, true, 0, 5.5);
00092     SetAt(loctestnum, loctesterr, vec, true, 1, 3.3);
00093     SetAt(loctestnum, loctesterr, vec, true, 2, 1.1);
00094
00095     GetFront(loctestnum, loctesterr, vec, true, 5.5);
00096     GetBack(loctestnum, loctesterr, vec, true, 1.1);
00097
00098     Exists(loctestnum, loctesterr, vec, true, 3.3);
00099
00100     FoldPreOrder(loctestnum, loctesterr, vec, true, &FoldAdd<double>, 0.0, 9.9);
00101     FoldPostOrder(loctestnum, loctesterr, vec, true, &FoldMultiply<double>, 1.0, 19.965);
00102   }
00103   catch (...) {
00104     loctestnum++; loctesterr++;
00105     cout « endl « "Unmanaged error! " « endl;
00106   }
00107   cout « "End of Vector<double> Test! (Errors/Tests: " « loctesterr « "/" « loctestnum « ")" « endl;
00108   testnum += loctestnum;
00109   testerr += loctesterr;
00110 }
00111
00112 void stestVectorString(uint & testnum, uint & testerr) {
00113   uint loctestnum = 0, loctesterr = 0;
00114   cout « endl « "Begin of Vector<string> Test:" « endl;
00115   try {
00116     lasd::SortableVector<string> vec(2);
00117
00118     Empty(loctestnum, loctesterr, vec, false);
00119     Size(loctestnum, loctesterr, vec, true, 2);
00120
00121     SetAt(loctestnum, loctesterr, vec, true, 0, string("A"));
00122     SetAt(loctestnum, loctesterr, vec, true, 1, string("B"));
00123
00124     GetFront(loctestnum, loctesterr, vec, true, string("A"));
00125     GetBack(loctestnum, loctesterr, vec, true, string("B"));
00126
00127     Exists(loctestnum, loctesterr, vec, true, string("A"));
00128
00129     MapPreOrder(loctestnum, loctesterr, vec, true, [](string & str) { MapStringAppend(str, string("
      "));  });
00130     TraversePreOrder(loctestnum, loctesterr, vec, true, &TraversePrint<string>);
00131     FoldPreOrder(loctestnum, loctesterr, vec, true, &FoldStringConcatenate, string("X"), string("XA B
      "));
00132     FoldPostOrder(loctestnum, loctesterr, vec, true, &FoldStringConcatenate, string("X"), string("XB A
      "));
00133
00134     Exists(loctestnum, loctesterr, vec, false, string("A"));
00135
00136     lasd::SortableVector<string> copvec(vec);
00137     EqualVector(loctestnum, loctesterr, vec, copvec, true);
00138     MapPreOrder(loctestnum, loctesterr, vec, true, [](string & str) { MapStringAppend(str,
      string("!"));  });
00139     NonEqualVector(loctestnum, loctesterr, vec, copvec, true);
00140
00141     copvec = move(vec);
00142     FoldPreOrder(loctestnum, loctesterr, copvec, true, &FoldStringConcatenate, string("?"), string("?A
      !B !"));
00143
00144     lasd::SortableVector<string> movvec(move(vec));
00145     FoldPreOrder(loctestnum, loctesterr, movvec, true, &FoldStringConcatenate, string("?"), string("?A
      B "));
00146     movvec.Sort();
00147     FoldPreOrder(loctestnum, loctesterr, movvec, true, &FoldStringConcatenate, string("?"), string("?A
      B "));
00148     SetAt(loctestnum, loctesterr, vec, false, 1, string(""));
00149     vec.Resize(1);
00150     SetAt(loctestnum, loctesterr, vec, true, 0, string("X"));
00151
00152     movvec.Clear();
00153     Empty(loctestnum, loctesterr, movvec, true);
00154   }
00155   catch (...) {
00156     loctestnum++; loctesterr++;
00157     cout « endl « "Unmanaged error! " « endl;
00158   }
00159   cout « "End of Vector<string> Test! (Errors/Tests: " « loctesterr « "/" « loctestnum « ")" « endl;
00160   testnum += loctestnum;
00161   testerr += loctesterr;
00162 }
```

```
00163
00164 void stestVector(uint & testnum, uint & testerr) {
00165   uint loctestnum = 0, loctesterr = 0;
00166   stestVectorInt(loctestnum, loctesterr);
00167   stestVectorDouble(loctestnum, loctesterr);
00168   stestVectorString(loctestnum, loctesterr);
00169   testnum += loctestnum;
00170   testerr += loctesterr;
00171   cout « endl « "Exercise 1A - Vector (Errors/Tests: " « loctesterr « "/" « loctestnum « ")" « endl;
00172 }
00173
00174 /* ************************************************************************** */
00175
00176 void stestListInt(uint & testnum, uint & testerr) {
00177   uint loctestnum = 0, loctesterr = 0;
00178   cout « endl « "Begin of List<int> Test:" « endl;
00179   try {
00180     lasd::List<int> lst;
00181     Empty(loctestnum, loctesterr, lst, true);
00182     Size(loctestnum, loctesterr, lst, true, 0);
00183
00184     GetFront(loctestnum, loctesterr, lst, false, 0);
00185     GetBack(loctestnum, loctesterr, lst, false, 0);
00186
00187     Exists(loctestnum, loctesterr, lst, false, 0);
00188
00189     TraversePreOrder(loctestnum, loctesterr, lst, true, &TraversePrint<int>);
00190     TraversePostOrder(loctestnum, loctesterr, lst, true, &TraversePrint<int>);
00191     FoldPreOrder(loctestnum, loctesterr, lst, true, &FoldAdd<int>, 0, 0);
00192     FoldPostOrder(loctestnum, loctesterr, lst, true, &FoldAdd<int>, 0, 0);
00193
00194     RemoveFromFront(loctestnum, loctesterr, lst, false);
00195     FrontNRemove(loctestnum, loctesterr, lst, false, 0);
00196
00197     InsertAtBack(loctestnum, loctesterr, lst, true, 4);
00198     InsertAtFront(loctestnum, loctesterr, lst, true, 5);
00199     InsertAtFront(loctestnum, loctesterr, lst, true, 9);
00200     InsertAtBack(loctestnum, loctesterr, lst, true, 2);
00201     InsertAtFront(loctestnum, loctesterr, lst, true, 1);
00202
00203     GetFront(loctestnum, loctesterr, lst, true, 1);
00204     GetBack(loctestnum, loctesterr, lst, true, 2);
00205     SetFront(loctestnum, loctesterr, lst, true, 2);
00206     SetBack(loctestnum, loctesterr, lst, true, 6);
00207
00208     GetAt(loctestnum, loctesterr, lst, true, 3, 4);
00209     SetAt(loctestnum, loctesterr, lst, true, 3, 3);
00210
00211     Exists(loctestnum, loctesterr, lst, false, 4);
00212
00213     TraversePreOrder(loctestnum, loctesterr, lst, true, &TraversePrint<int>);
00214     TraversePostOrder(loctestnum, loctesterr, lst, true, &TraversePrint<int>);
00215     FoldPreOrder(loctestnum, loctesterr, lst, true, &FoldAdd<int>, 0, 25);
00216     FoldPostOrder(loctestnum, loctesterr, lst, true, &FoldMultiply<int>, 1, 1620);
00217
00218     RemoveFromFront(loctestnum, loctesterr, lst, true);
00219     FrontNRemove(loctestnum, loctesterr, lst, true, 9);
00220     FoldPostOrder(loctestnum, loctesterr, lst, true, &FoldMultiply<int>, 1, 90);
00221
00222     lasd::List<int> coplst(lst);
00223     EqualList(loctestnum, loctesterr, lst, coplst, true);
00224     MapPreOrder(loctestnum, loctesterr, lst, true, &MapIncrement<int>);
00225     NonEqualList(loctestnum, loctesterr, lst, coplst, true);
00226
00227     InsertAtFront(loctestnum, loctesterr, lst, true, 0);
00228     InsertAtBack(loctestnum, loctesterr, lst, true, 0);
00229     NonEqualList(loctestnum, loctesterr, lst, coplst, true);
00230     coplst = lst;
00231     EqualList(loctestnum, loctesterr, lst, coplst, true);
00232
00233     RemoveFromFront(loctestnum, loctesterr, coplst, true);
00234     FrontNRemove(loctestnum, loctesterr, coplst, true, 6);
00235     coplst = move(lst);
00236     FoldPreOrder(loctestnum, loctesterr, lst, true, &FoldAdd<int>, 0, 11);
00237     FoldPreOrder(loctestnum, loctesterr, coplst, true, &FoldAdd<int>, 0, 17);
00238
00239     lasd::List<int> movlst(move(lst));
00240     MapPreOrder(loctestnum, loctesterr, movlst, true, &MapIncrement<int>);
00241     FoldPreOrder(loctestnum, loctesterr, movlst, true, &FoldAdd<int>, 0, 14);
00242
00243     InsertAtFront(loctestnum, loctesterr, movlst, true, 6);
00244     InsertAtBack(loctestnum, loctesterr, movlst, true, 8);
00245     RemoveFromFront(loctestnum, loctesterr, movlst, true);
00246     InsertAtBack(loctestnum, loctesterr, movlst, true, 7);
00247     FoldPreOrder(loctestnum, loctesterr, movlst, true, &FoldAdd<int>, 1, 30);
00248
00249     movlst.Clear();
```

```
00250     Empty(loctestnum, loctesterr, movlst, true);
00251     Size(loctestnum, loctesterr, movlst, true, 0);
00252   }
00253   catch (...) {
00254     loctestnum++; loctesterr++;
00255     cout « endl « "Unmanaged error! " « endl;
00256   }
00257   cout « "End of List<int> Test! (Errors/Tests: " « loctesterr « "/" « loctestnum « ")" « endl;
00258   testnum += loctestnum;
00259   testerr += loctesterr;
00260 }
00261
00262 void stestListDouble(uint & testnum, uint & testerr) {
00263   uint loctestnum = 0, loctesterr = 0;
00264   cout « endl « "Begin of List<double> Test:" « endl;
00265   try {
00266     lasd::List<double> lst;
00267     Empty(loctestnum, loctesterr, lst, true);
00268     Size(loctestnum, loctesterr, lst, true, 0);
00269
00270     InsertAtBack(loctestnum, loctesterr, lst, true, -2.5);
00271     InsertAtBack(loctestnum, loctesterr, lst, true, 2.5);
00272
00273     lst.Clear();
00274
00275     InsertAtBack(loctestnum, loctesterr, lst, true, 0.5);
00276     InsertAtFront(loctestnum, loctesterr, lst, true, 3.3);
00277     InsertAtFront(loctestnum, loctesterr, lst, true, 5.5);
00278     InsertAtBack(loctestnum, loctesterr, lst, true, 1.1);
00279
00280     GetFront(loctestnum, loctesterr, lst, true, 5.5);
00281     GetBack(loctestnum, loctesterr, lst, true, 1.1);
00282
00283     Exists(loctestnum, loctesterr, lst, false, 0.0);
00284
00285     TraversePreOrder(loctestnum, loctesterr, lst, true, &TraversePrint<double>);
00286     TraversePostOrder(loctestnum, loctesterr, lst, true, &TraversePrint<double>);
00287     FoldPreOrder(loctestnum, loctesterr, lst, true, &FoldAdd<double>, 0.0, 10.4);
00288     FoldPostOrder(loctestnum, loctesterr, lst, true, &FoldMultiply<double>, 1.0, 9.9825);
00289   }
00290   catch (...) {
00291     loctestnum++; loctesterr++;
00292     cout « endl « "Unmanaged error! " « endl;
00293   }
00294   cout « "End of List<double> Test! (Errors/Tests: " « loctesterr « "/" « loctestnum « ")" « endl;
00295   testnum += loctestnum;
00296   testerr += loctesterr;
00297 }
00298
00299 void stestListString(uint & testnum, uint & testerr) {
00300   uint loctestnum = 0, loctesterr = 0;
00301   cout « endl « "Begin of List<string> Test:" « endl;
00302   try {
00303     lasd::List<string> lst;
00304     Empty(loctestnum, loctesterr, lst, true);
00305     Size(loctestnum, loctesterr, lst, true, 0);
00306
00307     InsertAtFront(loctestnum, loctesterr, lst, true, string("A"));
00308     InsertAtBack(loctestnum, loctesterr, lst, true, string("B"));
00309
00310     GetFront(loctestnum, loctesterr, lst, true, string("A"));
00311     GetBack(loctestnum, loctesterr, lst, true, string("B"));
00312
00313     Exists(loctestnum, loctesterr, lst, true, string("B"));
00314
00315     MapPreOrder(loctestnum, loctesterr, lst, true, [](string & str) { MapStringAppend(str, string("
      A")); });
00316     TraversePreOrder(loctestnum, loctesterr, lst, true, &TraversePrint<string>);
00317     FoldPreOrder(loctestnum, loctesterr, lst, true, &FoldStringConcatenate, string("X"), string("XA B
      "));
00318     FoldPostOrder(loctestnum, loctesterr, lst, true, &FoldStringConcatenate, string("X"), string("XB A
      "));
00319
00320     Exists(loctestnum, loctesterr, lst, false, string("B"));
00321
00322     lasd::List<string> coplst(lst);
00323     EqualList(loctestnum, loctesterr, lst, coplst, true);
00324     RemoveFromFront(loctestnum, loctesterr, coplst, true);
00325     NonEqualList(loctestnum, loctesterr, lst, coplst, true);
00326
00327     lst = coplst;
00328     EqualList(loctestnum, loctesterr, lst, coplst, true);
00329     InsertAtBack(loctestnum, loctesterr, lst, true, string("A"));
00330     InsertAtFront(loctestnum, loctesterr, lst, true, string("C"));
00331     NonEqualList(loctestnum, loctesterr, lst, coplst, true);
00332
00333     coplst = move(lst);
```

```
00334     FoldPreOrder(loctestnum, loctesterr, coplst, true, &FoldStringConcatenate, string("?"),
      string("?CB A"));
00335   }
00336   catch (...) {
00337     loctestnum++; loctesterr++;
00338     cout « endl « "Unmanaged error! " « endl;
00339   }
00340   cout « "End of List<string> Test! (Errors/Tests: " « loctesterr « "/" « loctestnum « ")" « endl;
00341   testnum += loctestnum;
00342   testerr += loctesterr;
00343 }
00344
00345 void stestList(uint & testnum, uint & testerr) {
00346   uint loctestnum = 0, loctesterr = 0;
00347   stestListInt(loctestnum, loctesterr);
00348   stestListDouble(loctestnum, loctesterr);
00349   stestListString(loctestnum, loctesterr);
00350   testnum += loctestnum;
00351   testerr += loctesterr;
00352   cout « endl « "Exercise 1A - List (Errors/Tests: " « loctesterr « "/" « loctestnum « ")" « endl;
00353 }
00354
00355 /* ********************************************************************** */
00356
00357 void stestVectorListInt(uint & testnum, uint & testerr) {
00358   uint loctestnum = 0, loctesterr = 0;
00359   cout « endl « "Begin of Vector/List<int> Test:" « endl;
00360   try {
00361     lasd::SortableVector<int> vec(3);
00362     SetAt(loctestnum, loctesterr, vec, true, 0, -1);
00363     SetAt(loctestnum, loctesterr, vec, true, 1, 0);
00364     SetAt(loctestnum, loctesterr, vec, true, 2, 1);
00365
00366     lasd::List<int> lst;
00367     InsertAtFront(loctestnum, loctesterr, lst, true, 1);
00368     InsertAtFront(loctestnum, loctesterr, lst, true, 0);
00369     InsertAtFront(loctestnum, loctesterr, lst, true, -1);
00370
00371     EqualLinear(loctestnum, loctesterr, vec, lst, true);
00372
00373     lasd::SortableVector<int> copvec(lst);
00374     EqualVector(loctestnum, loctesterr, vec, copvec, true);
00375     lasd::SortableVector<int> copvecx(vec);
00376     EqualVector(loctestnum, loctesterr, copvecx, copvec, true);
00377
00378     lasd::List<int> coplst(vec);
00379     EqualList(loctestnum, loctesterr, lst, coplst, true);
00380     lasd::List<int> coplstx(lst);
00381     EqualList(loctestnum, loctesterr, coplstx, coplst, true);
00382   }
00383   catch (...) {
00384     loctestnum++; loctesterr++;
00385     cout « endl « "Unmanaged error! " « endl;
00386   }
00387   cout « "End of Vector/List<int> Test! (Errors/Tests: " « loctesterr « "/" « loctestnum « ")" « endl;
00388   testnum += loctestnum;
00389   testerr += loctesterr;
00390 }
00391
00392 void stestVectorListDouble(uint & testnum, uint & testerr) {
00393   uint loctestnum = 0, loctesterr = 0;
00394   cout « endl « "Begin of Vector/List<double> Test:" « endl;
00395   try {
00396     lasd::SortableVector<double> vec(3);
00397     SetAt(loctestnum, loctesterr, vec, true, 0, -0.5);
00398     SetAt(loctestnum, loctesterr, vec, true, 1, 0.0);
00399     SetAt(loctestnum, loctesterr, vec, true, 2, 0.5);
00400
00401     lasd::List<double> lst;
00402     InsertAtBack(loctestnum, loctesterr, lst, true, -0.5);
00403     InsertAtBack(loctestnum, loctesterr, lst, true, 0.0);
00404     InsertAtBack(loctestnum, loctesterr, lst, true, 0.5);
00405
00406     EqualLinear(loctestnum, loctesterr, vec, lst, true);
00407
00408     lasd::SortableVector<double> copvec(lst);
00409     EqualVector(loctestnum, loctesterr, vec, copvec, true);
00410     lasd::SortableVector<double> copvecx(vec);
00411     EqualVector(loctestnum, loctesterr, copvecx, copvec, true);
00412
00413     lasd::List<double> coplst(vec);
00414     EqualList(loctestnum, loctesterr, lst, coplst, true);
00415     lasd::List<double> coplstx(lst);
00416     EqualList(loctestnum, loctesterr, coplstx, coplst, true);
00417   }
00418   catch (...) {
00419     loctestnum++; loctesterr++;
```

```
00420     cout « endl « "Unmanaged error! " « endl;
00421   }
00422   cout « "End of Vector/List<double> Test! (Errors/Tests: " « loctesterr « "/" « loctestnum « ")" «
      endl;
00423   testnum += loctestnum;
00424   testerr += loctesterr;
00425 }
00426
00427 void stestVectorListString(uint & testnum, uint & testerr) {
00428   uint loctestnum = 0, loctesterr = 0;
00429   cout « endl « "Begin of Vector/List<string> Test:" « endl;
00430   try {
00431     lasd::SortableVector<string> vec(3);
00432     SetAt(loctestnum, loctesterr, vec, true, 0, string("A"));
00433     SetAt(loctestnum, loctesterr, vec, true, 1, string("B"));
00434     SetAt(loctestnum, loctesterr, vec, true, 2, string("C"));
00435
00436     lasd::List<string> lst;
00437     InsertAtFront(loctestnum, loctesterr, lst, true, string("B"));
00438     InsertAtBack(loctestnum, loctesterr, lst, true, string("C"));
00439     InsertAtFront(loctestnum, loctesterr, lst, true, string("A"));
00440
00441     EqualLinear(loctestnum, loctesterr, vec, lst, true);
00442
00443     lasd::SortableVector<string> copvec(lst);
00444     EqualVector(loctestnum, loctesterr, vec, copvec, true);
00445     lasd::SortableVector<string> copvecx(vec);
00446     EqualVector(loctestnum, loctesterr, copvecx, copvec, true);
00447
00448     lasd::List<string> coplst(vec);
00449     EqualList(loctestnum, loctesterr, lst, coplst, true);
00450     lasd::List<string> coplstx(lst);
00451     EqualList(loctestnum, loctesterr, coplstx, coplst, true);
00452
00453     Size(loctestnum, loctesterr, vec, true, 3);
00454     TraversePreOrder(loctestnum, loctesterr, vec, true, &TraversePrint<string>);
00455     Size(loctestnum, loctesterr, copvec, true, 3);
00456     TraversePreOrder(loctestnum, loctesterr, copvec, true, &TraversePrint<string>);
00457
00458     lasd::List<string> coplsty(move(vec));
00459     EqualList(loctestnum, loctesterr, coplst, coplsty, true);
00460     Size(loctestnum, loctesterr, vec, true, 3);
00461     TraversePreOrder(loctestnum, loctesterr, vec, true, &TraversePrint<string>);
00462     TraversePreOrder(loctestnum, loctesterr, copvec, true, &TraversePrint<string>);
00463     EqualVector(loctestnum, loctesterr, vec, copvec, false);
00464
00465     Size(loctestnum, loctesterr, lst, true, 3);
00466     TraversePreOrder(loctestnum, loctesterr, lst, true, &TraversePrint<string>);
00467     Size(loctestnum, loctesterr, coplst, true, 3);
00468     TraversePreOrder(loctestnum, loctesterr, coplst, true, &TraversePrint<string>);
00469
00470     lasd::SortableVector<string> copvecy(move(lst));
00471     EqualVector(loctestnum, loctesterr, copvec, copvecy, true);
00472     Size(loctestnum, loctesterr, lst, true, 3);
00473     TraversePreOrder(loctestnum, loctesterr, lst, true, &TraversePrint<string>);
00474     TraversePreOrder(loctestnum, loctesterr, coplst, true, &TraversePrint<string>);
00475     EqualList(loctestnum, loctesterr, lst, coplst, false);
00476   }
00477   catch (...) {
00478     loctestnum++; loctesterr++;
00479     cout « endl « "Unmanaged error! " « endl;
00480   }
00481   cout « "End of Vector/List<string> Test! (Errors/Tests: " « loctesterr « "/" « loctestnum « ")" «
      endl;
00482   testnum += loctestnum;
00483   testerr += loctesterr;
00484 }
00485
00486 void stestVectorList(uint & testnum, uint & testerr) {
00487   uint loctestnum = 0, loctesterr = 0;
00488   stestVectorListInt(loctestnum, loctesterr);
00489   stestVectorListDouble(loctestnum, loctesterr);
00490   stestVectorListString(loctestnum, loctesterr);
00491   testnum += loctestnum;
00492   testerr += loctesterr;
00493   cout « endl « "Exercise 1A - Vector/List (Errors/Tests: " « loctesterr « "/" « loctestnum « ")" «
      endl;
00494 }
00495
00496 /* ********************************************************************** */
00497
00498 void testSimpleExercise1A(uint & testnum, uint & testerr) {
00499   stestVector(testnum, testerr);
00500   stestList(testnum, testerr);
00501   stestVectorList(testnum, testerr);
00502   cout « endl « "Exercise 1A (Simple Test) (Errors/Tests: " « testerr « "/" « testnum « ")" « endl;
00503 }
```

7.55 Exercise1/zlasdtest/exercise1b/simpletest.cpp File Reference

```
#include <iostream>
#include "../container/container.hpp"
#include "../container/traversable.hpp"
#include "../container/testable.hpp"
#include "../container/dictionary.hpp"
#include "../container/linear.hpp"
#include "../vector/vector.hpp"
#include "../list/list.hpp"
#include "../set/set.hpp"
#include "../../set/lst/setlst.hpp"
#include "../../set/vec/setvec.hpp"
```

Functions

- void stestSetInt (lasd::Set< int > &set, uint &testnum, uint &testerr)
- void stestSetInt (uint &testnum, uint &testerr)
- void stestSetFloat (uint &testnum, uint &testerr)
- void stestSetString (lasd::Set< string > &set, uint &testnum, uint &testerr)
- void stestSetString (uint &testnum, uint &testerr)
- void testSimpleExercise1B (uint &testnum, uint &testerr)

7.55.1 Function Documentation

7.55.1.1 stestSetFloat()

```
void stestSetFloat (
            uint & testnum,
            uint & testerr)
```

Definition at line 254 of file simpletest.cpp.

7.55.1.2 stestSetInt() [1/2]

```
void stestSetInt (
            lasd::Set< int > & set,
            uint & testnum,
            uint & testerr)
```

Definition at line 25 of file simpletest.cpp.

7.55.1.3 stestSetInt() [2/2]

```
void stestSetInt (
            uint & testnum,
            uint & testerr)
```

Definition at line 120 of file simpletest.cpp.

7.55.1.4 testSetString() [1/2]

```
void testSetString (
              lasd::Set< string > & set,
              uint & testnum,
              uint & testerr)
```

Definition at line 356 of file simpletest.cpp.

7.55.1.5 testSetString() [2/2]

```
void testSetString (
              uint & testnum,
              uint & testerr)
```

Definition at line 372 of file simpletest.cpp.

7.55.1.6 testSimpleExercise1B()

```
void testSimpleExercise1B (
              uint & testnum,
              uint & testerr)
```

Definition at line 409 of file simpletest.cpp.

7.56 simpletest.cpp

Go to the documentation of this file.

```
00001
00002 #include <iostream>
00003
00004 /* ************************************************************************** */
00005
00006 #include "../container/container.hpp"
00007 #include "../container/traversable.hpp"
00008 #include "../container/testable.hpp"
00009 #include "../container/dictionary.hpp"
00010 #include "../container/linear.hpp"
00011
00012 #include "../vector/vector.hpp"
00013 #include "../list/list.hpp"
00014 #include "../set/set.hpp"
00015
00016 #include "../../set/lst/setlst.hpp"
00017 #include "../../set/vec/setvec.hpp"
00018
00019 /* ************************************************************************** */
00020
00021 using namespace std;
00022
00023 /* ************************************************************************** */
00024
00025 void stestSetInt(lasd::Set<int> & set, uint & testnum, uint & testerr) {
00026   uint loctestnum = 0, loctesterr = 0;
00027   try {
00028
00029     Empty(loctestnum, loctesterr, set, false);
00030     Size(loctestnum, loctesterr, set, true, 7);
00031
00032     GetAt(loctestnum, loctesterr, set, true, 0, 0);
00033
00034     TraversePreOrder(loctestnum, loctesterr, set, true, &TraversePrint<int>);
00035     TraversePostOrder(loctestnum, loctesterr, set, true, &TraversePrint<int>);
```

```
00036
00037     Min(loctestnum, loctesterr, set, true, 0);
00038     Max(loctestnum, loctesterr, set, true, 6);
00039
00040     RemoveMin(loctestnum, loctesterr, set, true);
00041     MinNRemove(loctestnum, loctesterr, set, true, 1);
00042
00043     InsertC(loctestnum, loctesterr, set, true, -1);
00044     InsertC(loctestnum, loctesterr, set, true, 1);
00045
00046     Min(loctestnum, loctesterr, set, true, -1);
00047     MaxNRemove(loctestnum, loctesterr, set, true, 6);
00048     Size(loctestnum, loctesterr, set, true, 6);
00049
00050     InsertC(loctestnum, loctesterr, set, true, 7);
00051
00052     Size(loctestnum, loctesterr, set, true, 7);
00053
00054     Max(loctestnum, loctesterr, set, true, 7);
00055
00056     InsertC(loctestnum, loctesterr, set, true, 8);
00057
00058     Size(loctestnum, loctesterr, set, true, 8);
00059
00060     Exists(loctestnum, loctesterr, set, true, 7);
00061     Exists(loctestnum, loctesterr, set, false, 9);
00062     Exists(loctestnum, loctesterr, set, false, 0);
00063     Exists(loctestnum, loctesterr, set, true, -1);
00064     Exists(loctestnum, loctesterr, set, true, 2);
00065
00066     TraversePreOrder(loctestnum, loctesterr, set, true, &TraversePrint<int>);
00067     TraversePostOrder(loctestnum, loctesterr, set, true, &TraversePrint<int>);
00068
00069     Remove(loctestnum, loctesterr, set, false, 6);
00070     Remove(loctestnum, loctesterr, set, true, 2);
00071
00072     TraversePreOrder(loctestnum, loctesterr, set, true, &TraversePrint<int>);
00073     TraversePostOrder(loctestnum, loctesterr, set, true, &TraversePrint<int>);
00074
00075     Exists(loctestnum, loctesterr, set, false, 6);
00076     Exists(loctestnum, loctesterr, set, false, 2);
00077
00078     RemoveMax(loctestnum, loctesterr, set, true);
00079     Max(loctestnum, loctesterr, set, true, 7);
00080
00081     TraversePreOrder(loctestnum, loctesterr, set, true, &TraversePrint<int>);
00082
00083     Predecessor(loctestnum, loctesterr, set, true, 4, 3);
00084     Predecessor(loctestnum, loctesterr, set, true, 5, 4);
00085
00086     Successor(loctestnum, loctesterr, set, true, 2, 3);
00087     Successor(loctestnum, loctesterr, set, true, 4, 5);
00088
00089     SuccessorNRemove(loctestnum, loctesterr, set, true, 0, 1);
00090     Min(loctestnum, loctesterr, set, true, -1);
00091
00092     TraversePreOrder(loctestnum, loctesterr, set, true, &TraversePrint<int>);
00093
00094     PredecessorNRemove(loctestnum, loctesterr, set, true, 7, 5);
00095     Max(loctestnum, loctesterr, set, true, 7);
00096
00097     TraversePostOrder(loctestnum, loctesterr, set, true, &TraversePrint<int>);
00098
00099     FoldPreOrder(loctestnum, loctesterr, set, true, &FoldAdd<int>, 0, 13);
00100     FoldPostOrder(loctestnum, loctesterr, set, true, &FoldAdd<int>, 0, 13);
00101
00102     TraversePreOrder(loctestnum, loctesterr, set, true, &TraversePrint<int>);
00103     TraversePostOrder(loctestnum, loctesterr, set, true, &TraversePrint<int>);
00104
00105     set.Clear();
00106
00107     Empty(loctestnum, loctesterr, set, true);
00108     Size(loctestnum, loctesterr, set, true, 0);
00109
00110   }
00111   catch (...) {
00112     loctestnum++; loctesterr++;
00113     cout « endl « "Unmanaged error! " « endl;
00114   }
00115   cout « "End of Set<int> Test! (Errors/Tests: " « loctesterr « "/" « loctestnum « ")" « endl;
00116   testnum += loctestnum;
00117   testerr += loctesterr;
00118 }
00119
00120 void stestSetInt(uint & testnum, uint & testerr) {
00121   uint loctestnum = 0, loctesterr = 0;
00122   cout « endl « "Begin of Set<int> Test" « endl;
```

```
00123   try {
00124     lasd::Vector<int> vec(7);
00125     SetAt(loctestnum, loctesterr, vec, true, 0, 3);
00126     SetAt(loctestnum, loctesterr, vec, true, 1, 1);
00127     SetAt(loctestnum, loctesterr, vec, true, 2, 6);
00128     SetAt(loctestnum, loctesterr, vec, true, 3, 5);
00129     SetAt(loctestnum, loctesterr, vec, true, 4, 0);
00130     SetAt(loctestnum, loctesterr, vec, true, 5, 2);
00131     SetAt(loctestnum, loctesterr, vec, true, 6, 4);
00132
00133     /* ********************************************************************** */
00134
00135     cout « endl « "Begin of SetVec<int> Test:" « endl;
00136     lasd::SetVec<int> setvec(vec);
00137     stestSetInt(setvec, loctestnum, loctesterr);
00138     cout « endl « "Begin of SetLst<int> Test:" « endl;
00139     lasd::SetLst<int> setlst(vec);
00140     stestSetInt(setlst, loctestnum, loctesterr);
00141     cout « "\n";
00142
00143     /* ********************************************************************** */
00144
00145     setvec.InsertAll(vec);
00146     lasd::SetVec<int> setvec1(setvec);
00147
00148     EqualSetVec(loctestnum, loctesterr, setvec, setvec1, true);
00149
00150     Remove(loctestnum, loctesterr, setvec1, true, 4);
00151
00152     NonEqualSetVec(loctestnum, loctesterr, setvec, setvec1, true);
00153
00154     InsertC(loctestnum, loctesterr, setvec1, true, 4);
00155
00156     EqualSetVec(loctestnum, loctesterr, setvec, setvec1, true);
00157
00158     lasd::SetVec<int> setvec2 = setvec1;
00159
00160     EqualSetVec(loctestnum, loctesterr, setvec1, setvec2, true);
00161
00162     RemovePredecessor(loctestnum, loctesterr, setvec1, true, 9);
00163
00164     EqualSetVec(loctestnum, loctesterr, setvec1, setvec2, false);
00165
00166     lasd::SetVec<int> setvec3(move(setvec2));
00167
00168     Empty(loctestnum, loctesterr, setvec2, true);
00169     Size(loctestnum, loctesterr, setvec2, true, 0);
00170
00171     Empty(loctestnum, loctesterr, setvec3, false);
00172     Size(loctestnum, loctesterr, setvec3, true, 7);
00173
00174     setvec2 = move(setvec1);
00175
00176     Empty(loctestnum, loctesterr, setvec1, true);
00177     Size(loctestnum, loctesterr, setvec1, true, 0);
00178
00179     Empty(loctestnum, loctesterr, setvec2, false);
00180     Size(loctestnum, loctesterr, setvec2, true, 6);
00181
00182     NonEqualSetVec(loctestnum, loctesterr, setvec3, setvec2, true);
00183
00184     Traverse(loctestnum, loctesterr, setvec2, true, &TraversePrint<int>);
00185     Traverse(loctestnum, loctesterr, setvec3, true, &TraversePrint<int>);
00186
00187     InsertC(loctestnum, loctesterr, setvec2, true, 6);
00188
00189     EqualSetVec(loctestnum, loctesterr, setvec3, setvec2, true);
00190
00191     /* ********************************************************************** */
00192
00193     setlst.InsertAll(vec);
00194     lasd::SetLst<int> setlst1(setlst);
00195
00196     EqualSetLst(loctestnum, loctesterr, setlst, setlst1, true);
00197
00198     Remove(loctestnum, loctesterr, setlst1, true, 4);
00199
00200     NonEqualSetLst(loctestnum, loctesterr, setlst, setlst1, true);
00201
00202     InsertC(loctestnum, loctesterr, setlst1, true, 4);
00203
00204     EqualSetLst(loctestnum, loctesterr, setlst, setlst1, true);
00205
00206     lasd::SetLst<int> setlst2 = setlst1;
00207
00208     EqualSetLst(loctestnum, loctesterr, setlst1, setlst2, true);
00209
```

```
00210     RemovePredecessor(loctestnum, loctesterr, setlst1, true, 9);
00211
00212     EqualSetLst(loctestnum, loctesterr, setlst1, setlst2, false);
00213
00214     lasd::SetLst<int> setlst3(move(setlst2));
00215
00216     Empty(loctestnum, loctesterr, setlst2, true);
00217     Size(loctestnum, loctesterr, setlst2, true, 0);
00218
00219     Empty(loctestnum, loctesterr, setlst3, false);
00220     Size(loctestnum, loctesterr, setlst3, true, 7);
00221
00222     setlst2 = move(setlst1);
00223
00224     Empty(loctestnum, loctesterr, setlst1, true);
00225     Size(loctestnum, loctesterr, setlst1, true, 0);
00226
00227     Empty(loctestnum, loctesterr, setlst2, false);
00228     Size(loctestnum, loctesterr, setlst2, true, 6);
00229
00230     NonEqualSetLst(loctestnum, loctesterr, setlst3, setlst2, true);
00231
00232     Traverse(loctestnum, loctesterr, setlst2, true, &TraversePrint<int>);
00233     Traverse(loctestnum, loctesterr, setlst3, true, &TraversePrint<int>);
00234
00235     InsertC(loctestnum, loctesterr, setlst2, true, 6);
00236
00237     EqualSetLst(loctestnum, loctesterr, setlst3, setlst2, true);
00238
00239     /* ************************************************************************ */
00240
00241     EqualLinear(loctestnum, loctesterr, setvec3, setlst2, true);
00242     NonEqualLinear(loctestnum, loctesterr, setlst3, setvec2, false);
00243
00244   }
00245   catch (...) {
00246     loctestnum++; loctesterr++;
00247     cout « endl « "Unmanaged error! " « endl;
00248   }
00249   cout « "End of Set<int> Test! (Errors/Tests: " « loctesterr « "/" « loctestnum « ")" « endl;
00250   testnum += loctestnum;
00251   testerr += loctesterr;
00252 }
00253
00254 void stestSetFloat(uint & testnum, uint & testerr) {
00255   uint loctestnum = 0, loctesterr = 0;
00256   cout « endl « "Begin of Set<double> Test" « endl;
00257   try {
00258     lasd::List<double> lst;
00259     InsertAtFront(loctestnum, loctesterr, lst, true, 4.0);
00260     InsertAtBack(loctestnum, loctesterr, lst, true, 0.4);
00261     InsertAtFront(loctestnum, loctesterr, lst, true, 1.2);
00262     InsertAtBack(loctestnum, loctesterr, lst, true, 2.1);
00263     InsertAtFront(loctestnum, loctesterr, lst, true, 3.5);
00264     InsertAtBack(loctestnum, loctesterr, lst, true, 5.3);
00265
00266     TraversePreOrder(loctestnum, loctesterr, lst, true, &TraversePrint<double>);
00267
00268     /* ************************************************************************ */
00269
00270     lasd::SetVec<double> setvec1(lst);
00271
00272     Empty(loctestnum, loctesterr, setvec1, false);
00273     Size(loctestnum, loctesterr, setvec1, true, 6);
00274
00275     TraversePreOrder(loctestnum, loctesterr, setvec1, true, &TraversePrint<double>);
00276     TraversePostOrder(loctestnum, loctesterr, setvec1, true, &TraversePrint<double>);
00277
00278     lasd::SetVec<double> setvec2;
00279
00280     InsertC(loctestnum, loctesterr, setvec2, true, 2.1);
00281     InsertC(loctestnum, loctesterr, setvec2, true, 0.4);
00282     InsertC(loctestnum, loctesterr, setvec2, true, 1.2);
00283     InsertC(loctestnum, loctesterr, setvec2, true, 3.5);
00284     InsertC(loctestnum, loctesterr, setvec2, true, 5.3);
00285     InsertC(loctestnum, loctesterr, setvec2, true, 4.0);
00286
00287     EqualSetVec(loctestnum, loctesterr, setvec1, setvec2, true);
00288     NonEqualSetVec(loctestnum, loctesterr, setvec1, setvec2, false);
00289
00290     setvec1.Clear();
00291     setvec2.Clear();
00292
00293     InsertC(loctestnum, loctesterr, setvec1, true, 0.2);
00294     InsertC(loctestnum, loctesterr, setvec1, true, 1.1);
00295     InsertC(loctestnum, loctesterr, setvec1, true, 2.1);
00296
```

```
00297     InsertC(loctestnum, loctesterr, setvec2, true, 2.1);
00298     InsertC(loctestnum, loctesterr, setvec2, true, 1.1);
00299     InsertC(loctestnum, loctesterr, setvec2, true, 0.2);
00300
00301     EqualSetVec(loctestnum, loctesterr, setvec1, setvec2, true);
00302     NonEqualSetVec(loctestnum, loctesterr, setvec1, setvec2, false);
00303
00304     /* ************************************************************************ */
00305
00306     lasd::SetLst<double> setlst1(lst);
00307
00308     Empty(loctestnum, loctesterr, setlst1, false);
00309     Size(loctestnum, loctesterr, setlst1, true, 6);
00310
00311     TraversePreOrder(loctestnum, loctesterr, setlst1, true, &TraversePrint<double>);
00312     TraversePostOrder(loctestnum, loctesterr, setlst1, true, &TraversePrint<double>);
00313
00314     lasd::SetLst<double> setlst2;
00315
00316     InsertC(loctestnum, loctesterr, setlst2, true, 2.1);
00317     InsertC(loctestnum, loctesterr, setlst2, true, 0.4);
00318     InsertC(loctestnum, loctesterr, setlst2, true, 1.2);
00319     InsertC(loctestnum, loctesterr, setlst2, true, 3.5);
00320     InsertC(loctestnum, loctesterr, setlst2, true, 5.3);
00321     InsertC(loctestnum, loctesterr, setlst2, true, 4.0);
00322
00323
00324     EqualSetLst(loctestnum, loctesterr, setlst1, setlst2, true);
00325     NonEqualSetLst(loctestnum, loctesterr, setlst1, setlst2, false);
00326
00327     setlst1.Clear();
00328     setlst2.Clear();
00329
00330     InsertC(loctestnum, loctesterr, setlst1, true, 0.2);
00331     InsertC(loctestnum, loctesterr, setlst1, true, 1.1);
00332     InsertC(loctestnum, loctesterr, setlst1, true, 2.1);
00333
00334     InsertC(loctestnum, loctesterr, setlst2, true, 2.1);
00335     InsertC(loctestnum, loctesterr, setlst2, true, 1.1);
00336     InsertC(loctestnum, loctesterr, setlst2, true, 0.2);
00337
00338     EqualSetLst(loctestnum, loctesterr, setlst1, setlst2, true);
00339     NonEqualSetLst(loctestnum, loctesterr, setlst1, setlst2, false);
00340
00341     /* ************************************************************************ */
00342
00343     EqualLinear(loctestnum, loctesterr, setvec1, setlst2, true);
00344     NonEqualLinear(loctestnum, loctesterr, setlst2, setvec2, false);
00345
00346   }
00347   catch (...) {
00348     loctestnum++; loctesterr++;
00349     cout « endl « "Unmanaged error! " « endl;
00350   }
00351   cout « "End of Set<double> Test! (Errors/Tests: " « loctesterr « "/" « loctestnum « ")" « endl;
00352   testnum += loctestnum;
00353   testerr += loctesterr;
00354 }
00355
00356 void stestSetString(lasd::Set<string> & set, uint & testnum, uint & testerr) {
00357   uint loctestnum = 0, loctesterr = 0;
00358   try {
00359     TraversePreOrder(loctestnum, loctesterr, set, true, &TraversePrint<string>);
00360     FoldPreOrder(loctestnum, loctesterr, set, true, &FoldStringConcatenate, string("?"),
      string("?ABCDE"));
00361     FoldPostOrder(loctestnum, loctesterr, set, true, &FoldStringConcatenate, string("?"),
      string("?EDCBA"));
00362   }
00363   catch (...) {
00364     loctestnum++; loctesterr++;
00365     cout « endl « "Unmanaged error! " « endl;
00366   }
00367   cout « "End of Set<string> Test! (Errors/Tests: " « loctesterr « "/" « loctestnum « ")" « endl;
00368   testnum += loctestnum;
00369   testerr += loctesterr;
00370 }
00371
00372 void stestSetString(uint & testnum, uint & testerr) {
00373   uint loctestnum = 0, loctesterr = 0;
00374   cout « endl « "Begin of Set<string> Test" « endl;
00375   try {
00376     lasd::Vector<string> vec(5);
00377     SetAt(loctestnum, loctesterr, vec, true, 0, string("C"));
00378     SetAt(loctestnum, loctesterr, vec, true, 1, string("A"));
00379     SetAt(loctestnum, loctesterr, vec, true, 2, string("E"));
00380     SetAt(loctestnum, loctesterr, vec, true, 3, string("D"));
00381     SetAt(loctestnum, loctesterr, vec, true, 4, string("B"));
```

```
00382
00383     /* ********************************************************************** */
00384
00385     cout « endl « "Begin of SetVec<string> Test:" « endl;
00386     lasd::SetVec<string> setvec(vec);
00387     stestSetString(setvec, loctestnum, loctesterr);
00388     cout « endl « "Begin of SetLst<string> Test:" « endl;
00389     lasd::SetLst<string> setlst(vec);
00390     stestSetString(setlst, loctestnum, loctesterr);
00391     cout « "\n";
00392
00393     /* ********************************************************************** */
00394
00395     EqualLinear(loctestnum, loctesterr, setvec, setlst, true);
00396
00397   }
00398   catch (...) {
00399     loctestnum++; loctesterr++;
00400     cout « endl « "Unmanaged error! " « endl;
00401   }
00402   cout « "End of Set<string> Test! (Errors/Tests: " « loctesterr « "/" « loctestnum « ")" « endl;
00403   testnum += loctestnum;
00404   testerr += loctesterr;
00405 }
00406
00407 /* ********************************************************************** */
00408
00409 void testSimpleExercise1B(uint & testnum, uint & testerr) {
00410   stestSetInt(testnum, testerr);
00411   stestSetFloat(testnum, testerr);
00412   stestSetString(testnum, testerr);
00413   cout « endl « "Exercise 1B (Simple Test) (Errors/Tests: " « testerr « "/" « testnum « ")" « endl;
00414 }
```

7.57 Exercise1/zlasdtest/exercise1a/test.hpp File Reference

Functions

- void testSimpleExercise1A (uint &, uint &)
- void testFullExercise1A (uint &, uint &)

7.57.1 Function Documentation

7.57.1.1 testFullExercise1A()

```
void testFullExercise1A (
              uint & ,
              uint & )
```

Definition at line 6 of file fulltest.cpp.

7.57.1.2 testSimpleExercise1A()

```
void testSimpleExercise1A (
              uint & testnum,
              uint & testerr)
```

Definition at line 498 of file simpletest.cpp.

7.58 test.hpp

Go to the documentation of this file.

```
00001
00002 #ifndef EX1ATEST_HPP
00003 #define EX1ATEST_HPP
00004
00005 /* ********************************************************************** */
00006
00007 void testSimpleExercise1A(uint &, uint &);
00008
00009 void testFullExercise1A(uint &, uint &);
00010
00011 /* ********************************************************************** */
00012
00013 #endif
```

7.59 Exercise1/zlasdtest/exercise1b/test.hpp File Reference

Functions

- void testSimpleExercise1B (uint &, uint &)
- void testFullExercise1B (uint &, uint &)

7.59.1 Function Documentation

7.59.1.1 testFullExercise1B()

```
void testFullExercise1B (
            uint & ,
            uint & )
```

Definition at line 6 of file fulltest.cpp.

7.59.1.2 testSimpleExercise1B()

```
void testSimpleExercise1B (
            uint & testnum,
            uint & testerr)
```

Definition at line 409 of file simpletest.cpp.

7.60 test.hpp

Go to the documentation of this file.

```
00001
00002 #ifndef EX1BTEST_HPP
00003 #define EX1BTEST_HPP
00004
00005 /* ********************************************************************** */
00006
00007 void testSimpleExercise1B(uint &, uint &);
00008
00009 void testFullExercise1B(uint &, uint &);
00010
00011 /* ********************************************************************** */
00012
00013 #endif
```

7.61 Exercise1/zlasdtest/test.hpp File Reference

Functions

- void lasdtest ()

7.61.1 Function Documentation

7.61.1.1 lasdtest()

```
void lasdtest ()
```

Definition at line 13 of file test.cpp.

7.62 test.hpp

Go to the documentation of this file.

```
00001
00002 #ifndef LASDTEST_HPP
00003 #define LASDTEST_HPP
00004
00005 /* ********************************************************************** */
00006
00007 void lasdtest();
00008
00009 /* ********************************************************************** */
00010
00011 #endif
```

7.63 Exercise1/zmytest/test.hpp File Reference

Functions

- void mytest ()

7.63.1 Function Documentation

7.63.1.1 mytest()

```
void mytest ()
```

Definition at line 12 of file test.cpp.

7.64 test.hpp

Go to the documentation of this file.

```
00001
00002 #ifndef MYTEST_HPP
00003 #define MYTEST_HPP
00004
00005 /* ************************************************************************** */
00006
00007 void mytest();
00008
00009 /* ************************************************************************** */
00010
00011 #endif
```

7.65 Exercise1/list/list.hpp File Reference

```
#include "../container/linear.hpp"
#include "list.cpp"
```

Classes

- class lasd::List< Data >
- struct lasd::List< Data >::Node

Namespaces

- namespace lasd

7.66 list.hpp

Go to the documentation of this file.

```
00001
00002 #ifndef LIST_HPP
00003 #define LIST_HPP
00004
00005 /* ************************************************************************** */
00006
00007 #include "../container/linear.hpp"
00008
00009 /* ************************************************************************** */
00010
00011 namespace lasd {
00012
00013 /* ************************************************************************** */
00014
00015 template <typename Data>
00016 class List {
00017   // Must extend MutableLinearContainer<Data>,
00018   //             ClearableContainer
00019
00020 private:
00021
00022   // ...
00023
00024 protected:
00025
00026   // using Container::???;
00027
00028   struct Node {
00029
00030     // Data
```

```
00031     // ...
00032
00033     /* ************************************************************************ */
00034
00035     // Specific constructors
00036     // ...
00037
00038     /* ************************************************************************ */
00039
00040     // Copy constructor
00041     // ...
00042
00043     // Move constructor
00044     // ...
00045
00046     /* ************************************************************************ */
00047
00048     // Destructor
00049     // ...
00050
00051     /* ************************************************************************ */
00052
00053     // Comparison operators
00054     // ...
00055
00056     /* ************************************************************************ */
00057
00058     // Specific member functions
00059
00060     // ...
00061
00062   };
00063
00064   // ...
00065
00066 public:
00067
00068   // Default constructor
00069   // List() specifiers;
00070
00071   /* ************************************************************************ */
00072
00073   // Specific constructor
00074   // List(argument) specifiers; // A list obtained from a TraversableContainer
00075   // List(argument) specifiers; // A list obtained from a MappableContainer
00076
00077   /* ************************************************************************ */
00078
00079   // Copy constructor
00080   // List(argument) specifiers;
00081
00082   // Move constructor
00083   // List(argument) specifiers;
00084
00085   /* ************************************************************************ */
00086
00087   // Destructor
00088   // ~List() specifiers;
00089
00090   /* ************************************************************************ */
00091
00092   // Copy assignment
00093   // type operator=(argument) specifiers;
00094
00095   // Move assignment
00096   // type operator=(argument) specifiers;
00097
00098   /* ************************************************************************ */
00099
00100   // Comparison operators
00101   // type operator==(argument) specifiers;
00102   // type operator!=(argument) specifiers;
00103
00104   /* ************************************************************************ */
00105
00106   // Specific member functions
00107
00108   // type InsertAtFront(argument) specifier; // Copy of the value
00109   // type InsertAtFront(argument) specifier; // Move of the value
00110   // type RemoveFromFront() specifier; // (must throw std::length_error when empty)
00111   // type FrontNRemove() specifier; // (must throw std::length_error when empty)
00112
00113   // type InsertAtBack(argument) specifier; // Copy of the value
00114   // type InsertAtBack(argument) specifier; // Move of the value
00115   // type RemoveFromBack() specifier; // (must throw std::length_error when empty)
00116   // type BackNRemove() specifier; // (must throw std::length_error when empty)
00117
```

```
00118   /* ************************************************************************ */
00119
00120   // Specific member functions (inherited from MutableLinearContainer)
00121
00122   // type operator[](argument) specifiers; // Override MutableLinearContainer member (must throw
      std::out_of_range when out of range)
00123
00124   // type Front() specifiers; // Override MutableLinearContainer member (must throw std::length_error
      when empty)
00125
00126   // type Back() specifiers; // Override MutableLinearContainer member (must throw std::length_error
      when empty)
00127
00128   /* ************************************************************************ */
00129
00130   // Specific member functions (inherited from LinearContainer)
00131
00132   // type operator[](argument) specifiers; // Override LinearContainer member (must throw
      std::out_of_range when out of range)
00133
00134   // type Front() specifiers; // Override LinearContainer member (must throw std::length_error when
      empty)
00135
00136   // type Back() specifiers; // Override LinearContainer member (must throw std::length_error when
      empty)
00137
00138   /* ************************************************************************ */
00139
00140   // Specific member function (inherited from MappableContainer)
00141
00142   // using typename MappableContainer<Data>::MapFun;
00143
00144   // type Map(argument) specifiers; // Override MappableContainer member
00145
00146   /* ************************************************************************ */
00147
00148   // Specific member function (inherited from PreOrderMappableContainer)
00149
00150   // type PreOrderMap(argument) specifiers; // Override PreOrderMappableContainer member
00151
00152   /* ************************************************************************ */
00153
00154   // Specific member function (inherited from PostOrderMappableContainer)
00155
00156   // type PostOrderMap(argument) specifiers; // Override PostOrderMappableContainer member
00157
00158   /* ************************************************************************ */
00159
00160   // Specific member function (inherited from TraversableContainer)
00161
00162   // using typename TraversableContainer<Data>::TraverseFun;
00163
00164   // type Traverse(arguments) specifiers; // Override TraversableContainer member
00165
00166   /* ************************************************************************ */
00167
00168   // Specific member function (inherited from PreOrderTraversableContainer)
00169
00170   // type PreOrderTraverse(arguments) specifiers; // Override PreOrderTraversableContainer member
00171
00172   /* ************************************************************************ */
00173
00174   // Specific member function (inherited from PostOrderTraversableContainer)
00175
00176   // type PostOrderTraverse(arguments) specifiers; // Override PostOrderTraversableContainer member
00177
00178   /* ************************************************************************ */
00179
00180   // Specific member function (inherited from ClearableContainer)
00181
00182   // type Clear() specifiers; // Override ClearableContainer member
00183
00184 protected:
00185
00186   // Auxiliary functions, if necessary!
00187
00188 };
00189
00190 /* ************************************************************************ */
00191
00192 }
00193
00194 #include "list.cpp"
00195
00196 #endif
```

7.67 Exercise1/zlasdtest/list/list.hpp File Reference

```
#include "../../list/list.hpp"
```

Functions

- template<typename Data>
  void InsertAtFront (uint &testnum, uint &testerr, lasd::List< Data > &lst, bool chk, const Data &val)
- template<typename Data>
  void RemoveFromFront (uint &testnum, uint &testerr, lasd::List< Data > &lst, bool chk)
- template<typename Data>
  void FrontNRemove (uint &testnum, uint &testerr, lasd::List< Data > &lst, bool chk, const Data &val)
- template<typename Data>
  void InsertAtBack (uint &testnum, uint &testerr, lasd::List< Data > &lst, bool chk, const Data &val)
- template<typename Data>
  void RemoveFromBack (uint &testnum, uint &testerr, lasd::List< Data > &lst, bool chk)
- template<typename Data>
  void BackNRemove (uint &testnum, uint &testerr, lasd::List< Data > &lst, bool chk, const Data &val)
- template<typename Data>
  void EqualList (uint &testnum, uint &testerr, const lasd::List< Data > &lst1, const lasd::List< Data > &lst2, bool chk)
- template<typename Data>
  void NonEqualList (uint &testnum, uint &testerr, const lasd::List< Data > &lst1, const lasd::List< Data > &lst2, bool chk)

7.67.1 Function Documentation

7.67.1.1 BackNRemove()

```
template<typename Data>
void BackNRemove (
            uint & testnum,
            uint & testerr,
            lasd::List< Data > & lst,
            bool chk,
            const Data & val)
```

Definition at line 96 of file list.hpp.

7.67.1.2 EqualList()

```
template<typename Data>
void EqualList (
            uint & testnum,
            uint & testerr,
            const lasd::List< Data > & lst1,
            const lasd::List< Data > & lst2,
            bool chk)
```

Definition at line 114 of file list.hpp.

7.67.1.3 FrontNRemove()

```
template<typename Data>
void FrontNRemove (
             uint & testnum,
             uint & testerr,
             lasd::List< Data > & lst,
             bool chk,
             const Data & val)
```

Definition at line 44 of file list.hpp.

7.67.1.4 InsertAtBack()

```
template<typename Data>
void InsertAtBack (
             uint & testnum,
             uint & testerr,
             lasd::List< Data > & lst,
             bool chk,
             const Data & val)
```

Definition at line 62 of file list.hpp.

7.67.1.5 InsertAtFront()

```
template<typename Data>
void InsertAtFront (
             uint & testnum,
             uint & testerr,
             lasd::List< Data > & lst,
             bool chk,
             const Data & val)
```

Definition at line 10 of file list.hpp.

7.67.1.6 NonEqualList()

```
template<typename Data>
void NonEqualList (
             uint & testnum,
             uint & testerr,
             const lasd::List< Data > & lst1,
             const lasd::List< Data > & lst2,
             bool chk)
```

Definition at line 128 of file list.hpp.

7.67.1.7 RemoveFromBack()

```
template<typename Data>
void RemoveFromBack (
             uint & testnum,
             uint & testerr,
             lasd::List< Data > & lst,
             bool chk)
```

Definition at line 77 of file list.hpp.

7.67.1.8 RemoveFromFront()

```
template<typename Data>
void RemoveFromFront (
             uint & testnum,
             uint & testerr,
             lasd::List< Data > & lst,
             bool chk)
```

Definition at line 25 of file list.hpp.

7.68 list.hpp

Go to the documentation of this file.

```
00001
00002 #ifndef LISTTEST_HPP
00003 #define LISTTEST_HPP
00004
00005 #include "../../list/list.hpp"
00006
00007 /* ********************************************************************** */
00008
00009 template <typename Data>
00010 void InsertAtFront(uint & testnum, uint & testerr, lasd::List<Data> & lst, bool chk, const Data & val)
      {
00011   bool tst;
00012   testnum++;
00013   try {
00014     std::cout « " " « testnum « " (" « testerr « ") Insert at the front of the list the value \"" «
      val « "\": ";
00015     lst.InsertAtFront(val);
00016     std::cout « ((tst = chk) ? "Correct" : "Error") « "!" « std::endl;
00017   }
00018   catch (std::exception & exc) {
00019     std::cout « "\"" « exc.what() « "\": " « ((tst = !chk) ? "Correct" : "Error") « "!" « std::endl;
00020   }
00021   testerr += (1 - (uint) tst);
00022 }
00023
00024 template <typename Data>
00025 void RemoveFromFront(uint & testnum, uint & testerr, lasd::List<Data> & lst, bool chk) {
00026   bool tst;
00027   testnum++;
00028   try {
00029     std::cout « " " « testnum « " (" « testerr « ") Remove from the list of \"" « lst.Front() « "\":
      ";
00030     lst.RemoveFromFront();
00031     std::cout « ((tst = chk) ? "Correct" : "Error") « "!" « std::endl;
00032   }
00033   catch (std::length_error & exc) {
00034     std::cout « exc.what() « "\": " « ((tst = !chk) ? "Correct" : "Error") « "!" « std::endl;
00035   }
00036   catch (std::exception & exc) {
00037     tst = false;
00038     std::cout « std::endl « "Wrong exception: " « exc.what() « "!" « std::endl;
00039   }
```

```
00040   testerr += (1 - (uint) tst);
00041 }
00042
00043 template <typename Data>
00044 void FrontNRemove(uint & testnum, uint & testerr, lasd::List<Data> & lst, bool chk, const Data & val)
      {
00045   bool tst;
00046   testnum++;
00047   try {
00048     std::cout « " " « testnum « " (" « testerr « ") FrontNRemove from the list of \"" « lst.Front() «
      "\": ";
00049     std::cout « ((tst = ((lst.FrontNRemove() == val) == chk)) ? "Correct" : "Error") « "!" «
      std::endl;
00050   }
00051   catch (std::length_error & exc) {
00052     std::cout « exc.what() « "\": " « ((tst = !chk) ? "Correct" : "Error") « "!" « std::endl;
00053   }
00054   catch (std::exception & exc) {
00055     tst = false;
00056     std::cout « std::endl « "Wrong exception: " « exc.what() « "!" « std::endl;
00057   }
00058   testerr += (1 - (uint) tst);
00059 }
00060
00061 template <typename Data>
00062 void InsertAtBack(uint & testnum, uint & testerr, lasd::List<Data> & lst, bool chk, const Data & val)
      {
00063   bool tst;
00064   testnum++;
00065   try {
00066     std::cout « " " « testnum « " (" « testerr « ") Insert at the back of the list the value \"" « val
      « "\": ";
00067     lst.InsertAtBack(val);
00068     std::cout « ((tst = chk) ? "Correct" : "Error") « "!" « std::endl;
00069   }
00070   catch (std::exception & exc) {
00071     std::cout « "\"" « exc.what() « "\": " « ((tst = !chk) ? "Correct" : "Error") « "!" « std::endl;
00072   }
00073   testerr += (1 - (uint) tst);
00074 }
00075
00076 template <typename Data>
00077 void RemoveFromBack(uint & testnum, uint & testerr, lasd::List<Data> & lst, bool chk) {
00078   bool tst;
00079   testnum++;
00080   try {
00081     std::cout « " " « testnum « " (" « testerr « ") Remove from the list of \"" « lst.Back() « "\": ";
00082     lst.RemoveFromBack();
00083     std::cout « ((tst = chk) ? "Correct" : "Error") « "!" « std::endl;
00084   }
00085   catch (std::length_error & exc) {
00086     std::cout « exc.what() « "\": " « ((tst = !chk) ? "Correct" : "Error") « "!" « std::endl;
00087   }
00088   catch (std::exception & exc) {
00089     tst = false;
00090     std::cout « std::endl « "Wrong exception: " « exc.what() « "!" « std::endl;
00091   }
00092   testerr += (1 - (uint) tst);
00093 }
00094
00095 template <typename Data>
00096 void BackNRemove(uint & testnum, uint & testerr, lasd::List<Data> & lst, bool chk, const Data & val) {
00097   bool tst;
00098   testnum++;
00099   try {
00100     std::cout « " " « testnum « " (" « testerr « ") BackNRemove from the list of \"" « lst.Back() «
      "\": ";
00101     std::cout « ((tst = ((lst.BackNRemove() == val) == chk)) ? "Correct" : "Error") « "!" « std::endl;
00102   }
00103   catch (std::length_error & exc) {
00104     std::cout « exc.what() « "\": " « ((tst = !chk) ? "Correct" : "Error") « "!" « std::endl;
00105   }
00106   catch (std::exception & exc) {
00107     tst = false;
00108     std::cout « std::endl « "Wrong exception: " « exc.what() « "!" « std::endl;
00109   }
00110   testerr += (1 - (uint) tst);
00111 }
00112
00113 template <typename Data>
00114 void EqualList(uint & testnum, uint & testerr, const lasd::List<Data> & lst1, const lasd::List<Data> &
      lst2, bool chk) {
00115   bool tst;
00116   testnum++;
00117   try {
00118     std::cout « " " « testnum « " (" « testerr « ") The two lists are " « ((tst = (lst1 == lst2)) ? ""
      : "not ") « "equal: ";
```

```
00119     std::cout « ((tst = (tst == chk)) ? "Correct" : "Error") « "!" « std::endl;
00120   }
00121   catch (std::exception & exc) {
00122     std::cout « "\"" « exc.what() « "\": " « ((tst = !chk) ? "Correct" : "Error") « "!" « std::endl;
00123   }
00124   testerr += (1 - (uint) tst);
00125 }
00126
00127 template <typename Data>
00128 void NonEqualList(uint & testnum, uint & testerr, const lasd::List<Data> & lst1, const
      lasd::List<Data> & lst2, bool chk) {
00129   bool tst;
00130   testnum++;
00131   try {
00132     std::cout « " " « testnum « " (" « testerr « ") The two lists are " « ((tst = (lst1 != lst2)) ?
      "not " : "") « "equal: ";
00133     std::cout « ((tst = (tst == chk)) ? "Correct" : "Error") « "!" « std::endl;
00134   }
00135   catch (std::exception & exc) {
00136     std::cout « "\"" « exc.what() « "\": " « ((tst = !chk) ? "Correct" : "Error") « "!" « std::endl;
00137   }
00138   testerr += (1 - (uint) tst);
00139 }
00140
00141 /* ********************************************************************** */
00142
00143 #endif
```

7.69 Exercise1/set/set.hpp File Reference

```
#include "../container/dictionary.hpp"
#include "../container/traversable.hpp"
```

Classes

- class lasd::Set< Data >

Namespaces

- namespace lasd

7.70 set.hpp

Go to the documentation of this file.

```
00001
00002 #ifndef SET_HPP
00003 #define SET_HPP
00004
00005 /* ********************************************************************** */
00006
00007 #include "../container/dictionary.hpp"
00008 #include "../container/traversable.hpp"
00009
00010 /* ********************************************************************** */
00011
00012 namespace lasd {
00013
00014 /* ********************************************************************** */
00015
00016 template <typename Data>
00017 class Set {
00018   // Must extend OrderedDictionaryContainer<Data>,
00019   //             LinearContainer<Data>,
00020   //             ClearableContainer
00021
```

```
00022 private:
00023
00024   // ...
00025
00026 protected:
00027
00028   // ...
00029
00030 public:
00031
00032   // Destructor
00033   // ~Set() specifiers
00034
00035   /* ********************************************************************** */
00036
00037   // Copy assignment
00038   // type operator=(argument); // Copy assignment of abstract types is not possible.
00039
00040   // Move assignment
00041   // type operator=(argument); // Move assignment of abstract types is not possible.
00042
00043 };
00044
00045 /* ********************************************************************** */
00046
00047 }
00048
00049 #endif
```

7.71 Exercise1/zlasdtest/set/set.hpp File Reference

```
#include "../../set/lst/setlst.hpp"
#include "../../set/vec/setvec.hpp"
```

Functions

- template<typename Data>
  void EqualSetLst (uint &testnum, uint &testerr, const lasd::SetLst< Data > &set1, const lasd::SetLst< Data > &set2, bool chk)
- template<typename Data>
  void NonEqualSetLst (uint &testnum, uint &testerr, const lasd::SetLst< Data > &set1, const lasd::SetLst< Data > &set2, bool chk)
- template<typename Data>
  void EqualSetVec (uint &testnum, uint &testerr, const lasd::SetVec< Data > &set1, const lasd::SetVec< Data > &set2, bool chk)
- template<typename Data>
  void NonEqualSetVec (uint &testnum, uint &testerr, const lasd::SetVec< Data > &set1, const lasd::SetVec< Data > &set2, bool chk)

7.71.1 Function Documentation

7.71.1.1 EqualSetLst()

```
template<typename Data>
void EqualSetLst (
            uint & testnum,
            uint & testerr,
            const lasd::SetLst< Data > & set1,
            const lasd::SetLst< Data > & set2,
            bool chk)
```

Definition at line 11 of file set.hpp.

7.71.1.2 EqualSetVec()

```
template<typename Data>
void EqualSetVec (
             uint & testnum,
             uint & testerr,
             const lasd::SetVec< Data > & set1,
             const lasd::SetVec< Data > & set2,
             bool chk)
```

Definition at line 41 of file set.hpp.

7.71.1.3 NonEqualSetLst()

```
template<typename Data>
void NonEqualSetLst (
             uint & testnum,
             uint & testerr,
             const lasd::SetLst< Data > & set1,
             const lasd::SetLst< Data > & set2,
             bool chk)
```

Definition at line 25 of file set.hpp.

7.71.1.4 NonEqualSetVec()

```
template<typename Data>
void NonEqualSetVec (
             uint & testnum,
             uint & testerr,
             const lasd::SetVec< Data > & set1,
             const lasd::SetVec< Data > & set2,
             bool chk)
```

Definition at line 55 of file set.hpp.

7.72 set.hpp

Go to the documentation of this file.

```
00001
00002 #ifndef SETTEST_HPP
00003 #define SETTEST_HPP
00004
00005 #include "../../set/lst/setlst.hpp"
00006 #include "../../set/vec/setvec.hpp"
00007
00008 /* ********************************************************************** */
00009
00010 template <typename Data>
00011 void EqualSetLst(uint & testnum, uint & testerr, const lasd::SetLst<Data> & set1, const
      lasd::SetLst<Data> & set2, bool chk) {
00012   bool tst;
00013   testnum++;
00014   try {
00015     std::cout « " " « testnum « " (" « testerr « ") The two setlsts are " « ((tst = (set1 == set2)) ?
      "" : "not ") « "equal: ";
00016     std::cout « ((tst = (tst == chk)) ? "Correct" : "Error") « "!" « std::endl;
00017   }
```

```
00018   catch (std::exception & exc) {
00019     std::cout « "\"" « exc.what() « "\": " « ((tst = !chk) ? "Correct" : "Error") « "!" « std::endl;
00020   }
00021   testerr += (1 - (uint) tst);
00022 }
00023
00024 template <typename Data>
00025 void NonEqualSetLst(uint & testnum, uint & testerr, const lasd::SetLst<Data> & set1, const
      lasd::SetLst<Data> & set2, bool chk) {
00026   bool tst;
00027   testnum++;
00028   try {
00029     std::cout « " " « testnum « " (" « testerr « ") The two setlsts are " « ((tst = (set1 != set2)) ?
      "not " : "") « "equal: ";
00030     std::cout « ((tst = (tst == chk)) ? "Correct" : "Error") « "!" « std::endl;
00031   }
00032   catch (std::exception & exc) {
00033     std::cout « "\"" « exc.what() « "\": " « ((tst = !chk) ? "Correct" : "Error") « "!" « std::endl;
00034   }
00035   testerr += (1 - (uint) tst);
00036 }
00037
00038 /* ********************************************************************** */
00039
00040 template <typename Data>
00041 void EqualSetVec(uint & testnum, uint & testerr, const lasd::SetVec<Data> & set1, const
      lasd::SetVec<Data> & set2, bool chk) {
00042   bool tst;
00043   testnum++;
00044   try {
00045     std::cout « " " « testnum « " (" « testerr « ") The two setvecs are " « ((tst = (set1 == set2)) ?
      "" : "not ") « "equal: ";
00046     std::cout « ((tst = (tst == chk)) ? "Correct" : "Error") « "!" « std::endl;
00047   }
00048   catch (std::exception & exc) {
00049     std::cout « "\"" « exc.what() « "\": " « ((tst = !chk) ? "Correct" : "Error") « "!" « std::endl;
00050   }
00051   testerr += (1 - (uint) tst);
00052 }
00053
00054 template <typename Data>
00055 void NonEqualSetVec(uint & testnum, uint & testerr, const lasd::SetVec<Data> & set1, const
      lasd::SetVec<Data> & set2, bool chk) {
00056   bool tst;
00057   testnum++;
00058   try {
00059     std::cout « " " « testnum « " (" « testerr « ") The two setvecs are " « ((tst = (set1 != set2)) ?
      "not " : "") « "equal: ";
00060     std::cout « ((tst = (tst == chk)) ? "Correct" : "Error") « "!" « std::endl;
00061   }
00062   catch (std::exception & exc) {
00063     std::cout « "\"" « exc.what() « "\": " « ((tst = !chk) ? "Correct" : "Error") « "!" « std::endl;
00064   }
00065   testerr += (1 - (uint) tst);
00066 }
00067
00068 /* ********************************************************************** */
00069
00070 #endif
```

7.73 Exercise1/zlasdtest/test.cpp File Reference

```
#include "./exercise1a/test.hpp"
#include "./exercise1b/test.hpp"
#include <iostream>
```

Functions

- void lasdtest ()

7.73.1 Function Documentation

7.73.1.1 lasdtest()

```
void lasdtest ()
```

Definition at line 13 of file test.cpp.

7.74 test.cpp

Go to the documentation of this file.

```
00001
00002 #include "./exercise1a/test.hpp"
00003 #include "./exercise1b/test.hpp"
00004
00005 /* ********************************************************************** */
00006
00007 #include <iostream>
00008
00009 using namespace std;
00010
00011 /* ********************************************************************** */
00012
00013 void lasdtest() {
00014   cout << endl << "~*~#~*~ Welcome to the LASD Test Suite ~*~#~*~ " << endl;
00015
00016   uint loctestnum, loctesterr;
00017   uint stestnum = 0, stesterr = 0;
00018
00019   loctestnum = 0; loctesterr = 0;
00020   testSimpleExercise1A(loctestnum, loctesterr);
00021   stestnum += loctestnum; stesterr += loctesterr;
00022
00023   loctestnum = 0; loctesterr = 0;
00024   testSimpleExercise1B(loctestnum, loctesterr);
00025   stestnum += loctestnum; stesterr += loctesterr;
00026
00027   cout << endl << "Exercise 1 (Simple Test) (Errors/Tests: " << stesterr << "/" << stestnum << ")";
00028
00029   cout << endl << "Goodbye!" << endl;
00030 }
```

7.75 Exercise1/zmytest/test.cpp File Reference

```
#include <iostream>
```

Functions

- void mytest ()

7.75.1 Function Documentation

7.75.1.1 mytest()

```
void mytest ()
```

Definition at line 12 of file test.cpp.

7.76 test.cpp

Go to the documentation of this file.

```
00001
00002 // #include "..."
00003
00004 /* ************************************************************************** */
00005
00006 #include <iostream>
00007
00008 using namespace std;
00009
00010 /* ************************************************************************** */
00011
00012 void mytest() {
00013   // ...
00014 }
```

7.77 Exercise1/vector/vector.hpp File Reference

```
#include "../container/linear.hpp"
#include "vector.cpp"
```

Classes

- class lasd::Vector< Data >

  Classe concreta che rappresenta un vettore dinamico di elementi.
- class lasd::SortableVector< Data >

  Classe concreta che estende Vector con capacità di ordinamento.

Namespaces

- namespace lasd

7.78 vector.hpp

Go to the documentation of this file.

```
00001 #ifndef VECTOR_HPP
00002 #define VECTOR_HPP
00003 #include "../container/linear.hpp"
00004
00005 namespace lasd
00006 {
00007
00015   template <typename Data>
00016   class Vector : virtual public MutableLinearContainer<Data>, virtual public ResizableContainer
00017   {
00018
00019   protected:
00020     using Container::size;
00021
00025     Data *elements = nullptr;
00026
00027   public:
00028     // Default constructor
00032     Vector() = default;
00033
00034     // Specific constructors
00035
00040     Vector(ulong);
```

```
00041
00046     Vector(const TraversableContainer<Data> &con);
00047
00052     Vector(MappableContainer<Data> &&con);
00053
00054     // Copy constructor
00055
00059     Vector(const Vector<Data> &);
00060
00061     // Move constructor
00062
00066     Vector(Vector<Data> &&) noexcept;
00067
00068     // Destructor
00069
00073     virtual ~Vector();
00074
00075     // Copy assignment
00079     inline Vector<Data> &operator=(const Vector<Data> &);
00080
00081     // Move assignment
00085     inline Vector<Data> &operator=(Vector<Data> &&) noexcept;
00086
00087     // Comparison operators
00088
00092     bool operator==(const Vector<Data> &) const noexcept;
00093
00097     inline bool operator!=(const Vector<Data> &) const noexcept;
00098
00099     // Override MutableLinearContainer members
00100
00101     inline Data &operator[](ulong) override;
00102     inline Data &Front() override;
00103     inline Data &Back() override;
00104
00105     // Override LinearContainer members
00106
00107     inline const Data &operator[](ulong) const override;
00108     inline const Data &Front() const override;
00109     inline const Data &Back() const override;
00110
00111     // Override ResizableContainer member
00112
00117     void Resize(ulong) override;
00118
00119     // Override ClearableContainer member
00120
00124     inline void Clear() override;
00125   };
00126
00132   template <typename Data>
00133   class SortableVector : public Vector<Data>, public SortableLinearContainer<Data>
00134   {
00135
00136   public:
00137     // Default constructor
00138     SortableVector() = default;
00139
00140     // Specific constructors
00141
00146     SortableVector(ulong dim) : Vector<Data>(dim) {}
00147
00152     SortableVector(const TraversableContainer<Data> &con) : Vector<Data>(con) {}
00153
00158     SortableVector(MappableContainer<Data> &&con) : Vector<Data>(std::move(con)) {}
00159
00160     // Copy constructor
00161     SortableVector(const SortableVector<Data> &other) : Vector<Data>(other) {}
00162
00163     // Move constructor
00164     SortableVector(SortableVector<Data> &&other) noexcept : Vector<Data>(std::move(other)) {}
00165
00166     // Destructor
00167     virtual ~SortableVector() = default;
00168
00169     // Copy assignment
00170     SortableVector<Data> &operator=(const SortableVector<Data> &other)
00171     {
00172       Vector<Data>::operator=(other);
00173       return *this;
00174     }
00175
00176     // Move assignment
00177     SortableVector<Data> &operator=(SortableVector<Data> &&other) noexcept
00178     {
00179       Vector<Data>::operator=(std::move(other));
00180       return *this;
```

```
00181     }
00182   };
00183
00184 }
00185
00186 #include "vector.cpp"
00187
00188 #endif
```

7.79 Exercise1/zlasdtest/vector/vector.hpp File Reference

```
#include "../../vector/vector.hpp"
```

Functions

- template<typename Data>
  void EqualVector (uint &testnum, uint &testerr, const lasd::Vector< Data > &vec1, const lasd::Vector< Data > &vec2, bool chk)
- template<typename Data>
  void NonEqualVector (uint &testnum, uint &testerr, const lasd::Vector< Data > &vec1, const lasd::Vector< Data > &vec2, bool chk)

7.79.1 Function Documentation

7.79.1.1 EqualVector()

```
template<typename Data>
void EqualVector (
            uint & testnum,
            uint & testerr,
            const lasd::Vector< Data > & vec1,
            const lasd::Vector< Data > & vec2,
            bool chk)
```

Definition at line 10 of file vector.hpp.

7.79.1.2 NonEqualVector()

```
template<typename Data>
void NonEqualVector (
            uint & testnum,
            uint & testerr,
            const lasd::Vector< Data > & vec1,
            const lasd::Vector< Data > & vec2,
            bool chk)
```

Definition at line 24 of file vector.hpp.

7.80 vector.hpp

Go to the documentation of this file.

```
00001
00002 #ifndef VECTORTEST_HPP
00003 #define VECTORTEST_HPP
00004
00005 #include "../../vector/vector.hpp"
00006
00007 /* ************************************************************************** */
00008
00009 template <typename Data>
00010 void EqualVector(uint & testnum, uint & testerr, const lasd::Vector<Data> & vec1, const
      lasd::Vector<Data> & vec2, bool chk) {
00011   bool tst;
00012   testnum++;
00013   try {
00014     std::cout « " " « testnum « " (" « testerr « ") The two vectors are " « ((tst = (vec1 == vec2)) ?
      "" : "not ") « "equal: ";
00015     std::cout « ((tst = (tst == chk)) ? "Correct" : "Error") « "!" « std::endl;
00016   }
00017   catch (std::exception & exc) {
00018     std::cout « "\"" « exc.what() « "\": " « ((tst = !chk) ? "Correct" : "Error") « "!" « std::endl;
00019   }
00020   testerr += (1 - (uint) tst);
00021 }
00022
00023 template <typename Data>
00024 void NonEqualVector(uint & testnum, uint & testerr, const lasd::Vector<Data> & vec1, const
      lasd::Vector<Data> & vec2, bool chk) {
00025   bool tst;
00026   testnum++;
00027   try {
00028     std::cout « " " « testnum « " (" « testerr « ") The two vectors are " « ((tst = (vec1 != vec2)) ?
      "not " : "") « "equal: ";
00029     std::cout « ((tst = (tst == chk)) ? "Correct" : "Error") « "!" « std::endl;
00030   }
00031   catch (std::exception & exc) {
00032     std::cout « "\"" « exc.what() « "\": " « ((tst = !chk) ? "Correct" : "Error") « "!" « std::endl;
00033   }
00034   testerr += (1 - (uint) tst);
00035 }
00036
00037 /* ************************************************************************** */
00038
00039 #endif
```

Index