HCL: Heterogeneous Container Library

Generated by Doxygen 1.8.9.1

Wed May 20 2015 17:08:56

Contents

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

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

Chapter 3

Class Documentation

3.1 heterogeneous::adaptor< container_type > Class Template Reference

Public Types

- typedef container_type::value_type **erased_type**
- template<typename native_typeA >
  using **iterator** = type_iterator< container_type, native_typeA >
- template<typename native_typeA >
  using **riterator** = type_reverse_iterator< container_type, native_typeA >

Public Member Functions

- template<typename native_typeA >
  iterator< native_typeA > **begin** ()
- template<typename native_typeA >
  riterator< native_typeA > **rbegin** ()
- template<typename native_typeA >
  iterator< native_typeA > **end** ()
- template<typename native_typeA >
  riterator< native_typeA > **rend** ()
- **adaptor** (container_type ∗object)
- **adaptor** (container_type &object)
- template<typename native_typeA >
  size_t size ()

  Returns number of elements of type native_typeA in object.
- template<typename native_typeA >
  bool empty ()

  Returns whether object contains any elements of type native_typeA or not.
- template<typename native_typeA >
  native_typeA & first ()

  Returns reference to first element of type native_typeA.
- template<typename native_typeA >
  native_typeA & last ()

  Returns reference to last element of type native_typeA.
- template<typename native_typeA >
  erased_type & at (const size_t &i)

  Returns reference to the type-erased object of the ith element of native type native_typeA.
- erased_type & at (const size_t &i)

Returns reference to the i-th type-erased object.

- template<typename native_typeA >
native_typeA & get (const size_t &i)

  Returns reference to ith element of type native_typeA.
- template<typename native_typeA , typename native_typeB >
bool swap (const size_t &i, const size_t &j)

  Swaps the positions of two elements of differing types.
- template<typename native_typeA >
bool swap (const size_t &i, const size_t &j)

  Swaps the positions of two elements of same types.

Friends

- template<typename container_t , typename native_t >
class **type_iterator**
- template<typename container_t , typename native_t >
class **type_reverse_iterator**

3.1.1 Member Function Documentation

3.1.1.1 template<typename container_type > template<typename native_typeA > erased_type& **heterogeneous::adaptor**< container_type >::at (const size_t & *i*) `[inline]`

Returns reference to the type-erased object of the ith element of native type native_typeA.

If the ith element of type native_typeA does not exsist, throws std::out_of_range exception.

Parameters

i	Index of the element of type native_typeA of which to return a reference to.

3.1.1.2 template<typename container_type > erased_type& **heterogeneous::adaptor**< container_type >::at (const size_t & *i*) `[inline]`

Returns reference to the i-th type-erased object.

If the ith element does not exsist, throws std::out_of_range exception.

Parameters

i	Index of the element of which to return a reference to.

3.1.1.3 template<typename container_type > template<typename native_typeA > native_typeA& **heterogeneous::adaptor**< container_type >::first ( ) `[inline]`

Returns reference to first element of type native_typeA.

If no elements of type native_typeA exist, throws std::out_of_range exception.

3.1.1.4 template<typename container_type > template<typename native_typeA > native_typeA& **heterogeneous::adaptor**< container_type >::get (const size_t & *i*) `[inline]`

Returns reference to ith element of type native_typeA.

If the ith element of type native_typeA does not exsist, throws std::out_of_range exception.

Parameters

i	Index of the element of type native_typeA of which to return a reference to.

3.1.1.5 template<typename container_type > template<typename native_typeA > native_typeA& heterogeneous::adaptor< container_type >::last ( ) [inline]

Returns reference to last element of type native_typeA.

If no elements of type native_typeA exist, throws std::out_of_range exception.

3.1.1.6 template<typename container_type > template<typename native_typeA , typename native_typeB > bool heterogeneous::adaptor< container_type >::swap (const size_t & *i*, const size_t & *j*) [inline]

Swaps the positions of two elements of differing types.

Finds the i-th element of type native_typeA and the j-th element of type native_typeB. If both exist, the positions are swapped, and function returns true. If i >= size<native_typeA>() or j >= size<native_typeB>(), no swap is performed and function returns false.

Parameters

i	Index of the element of type native_typeA to swap with j.
j	Index of the element of type native_typeB to swap with i.

3.1.1.7 template<typename container_type > template<typename native_typeA > bool heterogeneous::adaptor< container_type >::swap (const size_t & *i*, const size_t & *j*) [inline]

Swaps the positions of two elements of same types.

Finds the i-th and j-th elements of type native_typeA. If both exist, the positions are swapped, and function returns true. If i,j >= size<native_typeA>(), no swap is performed and function returns false.

Parameters

i	Index of the element of type native_typeA to swap with j.
j	Index of the element of type native_typeA to swap with i.

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

- adaptor.hpp

3.2 heterogeneous::heterodeque< T, Types...> Class Template Reference

Public Types

- typedef T **value_type**
- template<typename U >
  using **container_type** = std::deque< U >

Public Member Functions

- heterodeque< T, Types...> & **operator=** (const heterodeque< T, Types...> &x)
- bool operator== (const heterodeque< T, Types...> &rhs)

  Returns true if == operator evaluates to true for all elements in object.

- bool operator!= (const heterodeque< T, Types...> &rhs)

    Returns true if == operator evaluates to false for any element in object.
- bool operator< (const heterodeque< T, Types...> &rhs)

    Returns true if < operator evaluates to true for all elements in object.
- bool operator> (const heterodeque< T, Types...> &rhs)

    Returns true if > operator evaluates to true for all elements in object.
- bool operator<= (const heterodeque< T, Types...> &rhs)

    Returns true if <= operator evaluates to true for all elements in object.
- bool operator>= (const heterodeque< T, Types...> &rhs)

    Returns true if >= operator evaluates to true for all elements in object.
- bool eq (const heterodeque< T, Types...> &rhs)

    Same as operator==() but strictly enforces container element size matching.
- bool ne (const heterodeque< T, Types...> &rhs)

    Same as operator!=() but strictly enforces container element size matching.
- bool lt (const heterodeque< T, Types...> &rhs)

    Same as operator<() but strictly enforces container element size matching.
- bool gt (const heterodeque< T, Types...> &rhs)

    Same as operator>() but strictly enforces container element size matching.
- bool lte (const heterodeque< T, Types...> &rhs)

    Same as operator<=() but strictly enforces container element size matching.
- bool gte (const heterodeque< T, Types...> &rhs)

    Same as operator>=() but strictly enforces container element size matching.
- template<typename U , size_t N = 0>
  container_type< U >::iterator **begin** ()
- template<typename U , size_t N = 0>
  container_type< U >::iterator **end** ()
- template<typename U , size_t N = 0>
  container_type< U >::reverse_iterator **rbegin** ()
- template<typename U , size_t N = 0>
  container_type< U >::reverse_iterator **rend** ()
- template<typename U , size_t N = 0>
  container_type< U >::const_iterator **cbegin** ()
- template<typename U , size_t N = 0>
  container_type< U >::const_iterator **cend** ()
- template<typename U , size_t N = 0>
  container_type< U >::const_reverse_iterator **crbegin** ()
- template<typename U , size_t N = 0>
  container_type< U >::const_reverse_iterator **crend** ()
- size_t size ()

    Returns the total number of elements in object.
- template<typename U , size_t N = 0>
  size_t size () const

    Returns number of elements for the Nth container of type U.
- template<typename U , size_t N = 0>
  size_t max_size ()

    Returns the maximum number of elements the Nth container of type U can hold.
- template<typename U , size_t N = 0>
  void resize (size_t n)

    Resizes the Nth container of type U so that it contains at least n elements.
- template<typename U , size_t N = 0>
  void resize (size_t n, const U &val)

    Resizes the Nth container of type U so that it contains at least n elements.

- template<typename U , size_t N = 0>
  bool empty ()

  Resizes whether the Nth container of type U is empty.
- template<typename U , size_t N = 0>
  void shrink_to_fit ()

  Resizes the Nth container of type U reduce its capacity to fit its size.
- template<typename U >
  bool contains ()

  Returns whether object stores type U.
- template<typename U >
  size_t multiplicity ()

  Returns the number of containers with type U.
- template<typename U , size_t N = 0>
  U & at (size_t n)

  Returns reference to the nth element of the Nth container of type U.
- template<typename U , size_t N = 0>
  const U & at (size_t n) const

  Returns const reference to the nth element of the Nth container of type U.
- template<typename U , size_t N = 0>
  U & front ()

  Returns reference to the first element of the Nth container of type U.
- template<typename U , size_t N = 0>
  const U & front () const

  Returns const reference to the first element of the Nth container of type U.
- template<typename U , size_t N = 0>
  U & back ()

  Returns reference to the last element of the Nth container of type U.
- template<typename U , size_t N = 0>
  const U & back () const

  Returns const reference to the last element of the Nth container of type U.
- template<typename U , size_t N = 0>
  container_type< U > * container ()

  Returns reference to the Nth container of type U.
- template<typename U , size_t N = 0>
  const container_type< U > * container () const

  Returns const reference to the Nth container of type U.
- template<size_t N = 0>
  std::type_index type ()

  Returns std::type_index of items within the Nth container in object.
- template<typename U , size_t N = 0, typename InputIterator >
  void assign (InputIterator first, InputIterator last)

  Assigns new contents to the Nth container of type U, replacing its current contents, and modifying its size accordingly.
- template<typename U , size_t N = 0>
  void assign (size_t n, const U &val)

  Assigns new contents to the Nth container of type U, replacing its current contents, and modifying its size accordingly.
- template<typename U , size_t N = 0>
  void assign (std::initializer_list< U > il)

  Assigns new contents to the Nth container of type U, replacing its current contents, and modifying its size accordingly.
- template<typename U , size_t N = 0, typename V >
  void push_back (const V &val)

  Adds val as the last element of the Nth container of type U.
- template<typename U , size_t N = 0, typename V >
  void push_back (V &&val)

Adds val as the last element of the Nth container of type U.

- template<typename U , size_t N = 0, typename V >
  void push_front (const V &val)

  Adds val as the first element of the Nth container of type U.
- template<typename U , size_t N = 0, typename V >
  void push_front (V &&val)

  Adds val as the first element of the Nth container of type U.
- template<typename U , size_t N = 0>
  void pop_back ()

  Removes the last element of the Nth container of type U.
- template<typename U , size_t N = 0>
  void pop_front ()

  Removes the first element of the Nth container of type U.
- template<typename U , size_t N = 0>
  void insert (typename container_type< U >::const_iterator position, const U &val)

  Inserts val into the Nth container of type U at the location specified by position.
- template<typename U , size_t N = 0>
  void insert (typename container_type< U >::const_iterator position, U &&val)

  Inserts val into the Nth container of type U at the location specified by position.
- template<typename U , size_t N = 0>
  void insert (typename container_type< U >::const_iterator position, size_t n, const U &val)

  Inserts n copies of val into the Nth container of type U, starting at the location specified by position.
- template<typename U , size_t N = 0, typename InputIterator >
  void insert (typename container_type< U >::const_iterator position, InputIterator first, InputIterator last)

  Inserts copies of the values between [first,last) into the Nth container of type U, starting at the location specified by position.
- template<typename U , size_t N = 0>
  void insert (typename container_type< U >::const_iterator position, std::initializer_list< U > il)

  Inserts il into the Nth container of type U, starting at the location specified by position.
- template<typename U , size_t N = 0>
  void erase (typename container_type< U >::const_iterator position)

  Removes the element specified by position in the Nth container of type U.
- template<typename U , size_t N = 0>
  void erase (typename container_type< U >::const_iterator first, typename container_type< U >::const_↩
  iterator last)

  Removes the range of elements between locations specified by first and last in the Nth container of type U.
- template<typename U , size_t N = 0>
  void swap (container_type< U > &x)

  Swaps the contents of the Nth container of type U with that of x.
- void swap (heterodeque< T, Types...> &x)

  Swaps contents of object with x.
- template<typename U , size_t N = 0>
  void clear ()

  Erases the of contents the Nth container of type U.
- template<typename U , size_t N = 0, typename... Args>
  void emplace (typename container_type< U >::const_iterator position, Args &&...args)

  The Nth container of type U is extended by inserting a new element at position.
- template<typename U , size_t N = 0, typename... Args>
  void emplace_back (Args &&...args)

  The Nth container of type U is extended by inserting a new element after its current last element.
- template<typename U , size_t N = 0, typename... Args>
  void emplace_front (Args &&...args)

  The Nth container of type U is extended by inserting a new element after its current last element.

- template<typename U , size_t N = 0>
  container_type< U > & set (const container_type< U > &x)

  Sets the contents of the Nth container of type U to those in x.
- template<typename U , size_t N = 0>
  container_type< U > & set (container_type< U > &&x)

  Sets the contents of the Nth container of type U to those in x.
- template<typename U , size_t N = 0>
  container_type< U > & set (std::initializer_list< U > il)

  Sets the contents of the Nth container of type U to il.
- template<typename U , size_t N = 0>
  container_type< U >::allocator_type get_allocator () const

  Returns a copy of the allocator object associated with the Nth container of type U.

Friends

- template<typename... Args>
  class **heterodeque**

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

- heterodeque.hpp

3.3 heterogeneous::heterovector< T, Types...> Class Template Reference

Public Types

- typedef T **value_type**
- template<typename U >
  using **container_type** = std::vector< U >

Public Member Functions

- heterovector< T, Types...> & **operator=** (const heterovector< T, Types...> &x)
- bool operator== (const heterovector< T, Types...> &rhs)

  Returns true if == operator evaluates to true for each element in object.
- bool operator!= (const heterovector< T, Types...> &rhs)

  Returns true if == operator evaluates to false for any element in object.
- bool operator< (const heterovector< T, Types...> &rhs)

  Returns true if < operator evaluates to true for each element in object.
- bool operator> (const heterovector< T, Types...> &rhs)

  Returns true if > operator evaluates to true for each element in object.
- bool operator<= (const heterovector< T, Types...> &rhs)

  Returns true if <= operator evaluates to true for each element in object.
- bool operator>= (const heterovector< T, Types...> &rhs)

  Returns true if >= operator evaluates to true for each element in object.
- bool eq (const heterovector< T, Types...> &rhs)

  Same as operator==() but strictly enforces container element size matching.
- bool ne (const heterovector< T, Types...> &rhs)

  Same as operator!=() but strictly enforces container element size matching.
- bool lt (const heterovector< T, Types...> &rhs)

  Same as operator<() but strictly enforces container element size matching.

- bool gt (const heterovector< T, Types...> &rhs)

  Same as operator>() but strictly enforces container element size matching.
- bool lte (const heterovector< T, Types...> &rhs)

  Same as operator<=() but strictly enforces container element size matching.
- bool gte (const heterovector< T, Types...> &rhs)

  Same as operator>=() but strictly enforces container element size matching.
- template<typename U , size_t N = 0>
  container_type< U >::iterator **begin** ()
- template<typename U , size_t N = 0>
  container_type< U >::iterator **end** ()
- template<typename U , size_t N = 0>
  container_type< U >::reverse_iterator **rbegin** ()
- template<typename U , size_t N = 0>
  container_type< U >::reverse_iterator **rend** ()
- template<typename U , size_t N = 0>
  container_type< U >::const_iterator **cbegin** ()
- template<typename U , size_t N = 0>
  container_type< U >::const_iterator **cend** ()
- template<typename U , size_t N = 0>
  container_type< U >::const_reverse_iterator **crbegin** ()
- template<typename U , size_t N = 0>
  container_type< U >::const_reverse_iterator **crend** ()
- size_t size ()

  Returns the total number of elements in object.
- template<typename U , size_t N = 0>
  size_t size () const

  Returns number of elements for the Nth container of type U.
- template<typename U , size_t N = 0>
  size_t max_size ()

  Returns the maximum number of elements the Nth container of type U can hold.
- template<typename U , size_t N = 0>
  void resize (size_t n)

  Resizes the Nth container of type U so that it contains at least n elements.
- template<typename U , size_t N = 0>
  void resize (size_t n, const U &val)

  Resizes the Nth container of type U so that it contains at least n elements.
- template<typename U , size_t N = 0>
  size_t capacity ()

  Returns the number of elements the Nth container of type U can store before reallocation.
- template<typename U , size_t N = 0>
  bool empty ()

  Resizes whether the Nth container of type U is empty.
- template<typename U , size_t N = 0>
  void reserve (size_t n)

  Requests the capacity of Nth container of type U is enough to contain at least n elements.
- template<typename U , size_t N = 0>
  void shrink_to_fit ()

  Resizes the Nth container of type U reduce its capacity to fit its size.
- template<typename U >
  bool contains ()

  Returns whether object stores type U.
- template<typename U >
  size_t multiplicity ()

Returns the number of containers with type U.

- template<typename U , size_t N = 0>
  U & at (size_t n)

  Returns reference to the nth element of the Nth container of type U.
- template<typename U , size_t N = 0>
  const U & at (size_t n) const

  Returns const reference to the nth element of the Nth container of type U.
- template<typename U , size_t N = 0>
  U & front ()

  Returns reference to the first element of the Nth container of type U.
- template<typename U , size_t N = 0>
  const U & front () const

  Returns const reference to the first element of the Nth container of type U.
- template<typename U , size_t N = 0>
  U & back ()

  Returns reference to the last element of the Nth container of type U.
- template<typename U , size_t N = 0>
  const U & back () const

  Returns const reference to the last element of the Nth container of type U.
- template<typename U , size_t N = 0>
  U ∗ data ()

  Returns pointer to the data contained in the Nth container of type U.
- template<typename U , size_t N = 0>
  const U ∗ data () const

  Returns const pointer to the data contained in the Nth container of type U.
- template<typename U , size_t N = 0>
  container_type< U > ∗ container ()

  Returns reference to the Nth container of type U.
- template<typename U , size_t N = 0>
  const container_type< U > ∗ container () const

  Returns const reference to the Nth container of type U.
- template<size_t N = 0>
  std::type_index type ()

  Returns std::type_index of items within the Nth container in object.
- template<typename U , size_t N = 0, typename InputIterator >
  void assign (InputIterator first, InputIterator last)

  Assigns new contents to the Nth container of type U, replacing its current contents, and modifying its size accordingly.
- template<typename U , size_t N = 0>
  void assign (size_t n, const U &val)

  Assigns new contents to the Nth container of type U, replacing its current contents, and modifying its size accordingly.
- template<typename U , size_t N = 0>
  void assign (std::initializer_list< U > il)

  Assigns new contents to the Nth container of type U, replacing its current contents, and modifying its size accordingly.
- template<typename U , size_t N = 0, typename V >
  void push_back (const V &val)

  Adds val as the last element of the Nth container of type U.
- template<typename U , size_t N = 0, typename V >
  void push_back (V &&val)

  Adds val as the last element of the Nth container of type U.
- template<typename U , size_t N = 0>
  void pop_back ()

  Removes the last element of the Nth container of type U.

- template<typename U , size_t N = 0>
  void insert (typename container_type< U >::const_iterator position, const U &val)

  Inserts val into the Nth container of type U at the location specified by position.
- template<typename U , size_t N = 0>
  void insert (typename container_type< U >::const_iterator position, U &&val)

  Inserts val into the Nth container of type U at the location specified by position.
- template<typename U , size_t N = 0>
  void insert (typename container_type< U >::const_iterator position, size_t n, const U &val)

  Inserts n copies of val into the Nth container of type U, starting at the location specified by position.
- template<typename U , size_t N = 0, typename InputIterator >
  void insert (typename container_type< U >::const_iterator position, InputIterator first, InputIterator last)

  Inserts copies of the values between [first,last) into the Nth container of type U, starting at the location specified by position.
- template<typename U , size_t N = 0>
  void insert (typename container_type< U >::const_iterator position, std::initializer_list< U > il)

  Inserts il into the Nth container of type U, starting at the location specified by position.
- template<typename U , size_t N = 0>
  void erase (typename container_type< U >::const_iterator position)

  Removes the element specified by position in the Nth container of type U.
- template<typename U , size_t N = 0>
  void erase (typename container_type< U >::const_iterator first, typename container_type< U >::const_↩
  iterator last)

  Removes the range of elements between locations specified by first and last in the Nth container of type U.
- template<typename U , size_t N = 0>
  void swap (container_type< U > &x)

  Swaps the contents of the Nth container of type U with that of x.
- void swap (heterovector< T, Types...> &x)

  Swaps contents of object with x.
- template<typename U , size_t N = 0>
  void clear ()

  Erases the of contents the Nth container of type U.
- template<typename U , size_t N = 0, typename... Args>
  void emplace (typename container_type< U >::const_iterator position, Args &&...args)

  The Nth container of type U is extended by inserting a new element at position.
- template<typename U , size_t N = 0, typename... Args>
  void emplace_back (Args &&...args)

  The Nth container of type U is extended by inserting a new element after its current last element.
- template<typename U , size_t N = 0>
  container_type< U > & set (const container_type< U > &x)

  Sets the contents of the Nth container of type U to those in x.
- template<typename U , size_t N = 0>
  container_type< U > & set (container_type< U > &&x)

  Sets the contents of the Nth container of type U to those in x.
- template<typename U , size_t N = 0>
  container_type< U > & set (std::initializer_list< U > il)

  Sets the contents of the Nth container of type U to il.
- template<typename U , size_t N = 0>
  container_type< U >::allocator_type get_allocator () const

  Returns a copy of the allocator object associated with the Nth container of type U.

Friends

- template<typename... Args>
  class **heterovector**

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

- heterovector.hpp

Chapter 4

File Documentation

4.1 adaptor.hpp File Reference

```
#include <assert.h>
#include <stdexcept>
#include <typeinfo>
#include <typeindex>
#include <string>
#include <boost\any.hpp>
```

Classes

- class heterogeneous::adaptor< container_type >

Functions

- template<typename container_type >
  container_type & **boost::get** (boost::any &a)

  get<container_type> overload for boost::any.

4.1.1 Detailed Description

This templated class defines an interface for retrieving and modifying the contents of a container of any type-erased data type (boost::any, boost::variant, etc)

4.2 heterodeque.hpp File Reference

```
#include <stdexcept>
#include <typeinfo>
#include <typeindex>
#include <deque>
#include <string>
```

Classes

- class heterogeneous::heterodeque< T, Types...>

Functions

- template<typename... Args_lhs, typename... Args_rhs>
  bool **heterogeneous::operator==** (const heterodeque< Args_lhs...> &lhs, const heterodeque< Args_↩
  rhs...> &rhs)
- template<typename... Args_lhs, typename... Args_rhs>
  bool **heterogeneous::operator!=** (const heterodeque< Args_lhs...> &lhs, const heterodeque< Args_↩
  rhs...> &rhs)
- template<typename... Args_lhs, typename... Args_rhs>
  bool **heterogeneous::operator**< (const heterodeque< Args_lhs...> &lhs, const heterodeque< Args_↩
  rhs...> &rhs)
- template<typename... Args_lhs, typename... Args_rhs>
  bool **heterogeneous::operator<=** (const heterodeque< Args_lhs...> &lhs, const heterodeque< Args_↩
  rhs...> &rhs)
- template<typename... Args_lhs, typename... Args_rhs>
  bool **heterogeneous::operator**> (const heterodeque< Args_lhs...> &lhs, const heterodeque< Args_↩
  rhs...> &rhs)
- template<typename... Args_lhs, typename... Args_rhs>
  bool **heterogeneous::operator>=** (const heterodeque< Args_lhs...> &lhs, const heterodeque< Args_↩
  rhs...> &rhs)

4.2.1 Detailed Description

This templated class replicates the features of an std::deque that can handle heterogeneous data types.

4.3 heterovector.hpp File Reference

```
#include <stdexcept>
#include <typeinfo>
#include <typeindex>
#include <vector>
#include <string>
```

Classes

- class heterogeneous::heterovector< T, Types...>

Functions

- template<typename... Args_lhs, typename... Args_rhs>
  bool **heterogeneous::operator==** (const heterovector< Args_lhs...> &lhs, const heterovector< Args_↩
  rhs...> &rhs)
- template<typename... Args_lhs, typename... Args_rhs>
  bool **heterogeneous::operator!=** (const heterovector< Args_lhs...> &lhs, const heterovector< Args_rhs...>
  &rhs)
- template<typename... Args_lhs, typename... Args_rhs>
  bool **heterogeneous::operator**< (const heterovector< Args_lhs...> &lhs, const heterovector< Args_rhs...>
  &rhs)

- template<typename... Args_lhs, typename... Args_rhs>
  bool **heterogeneous::operator<=** (const heterovector< Args_lhs...> &lhs, const heterovector< Args_↩ rhs...> &rhs)
- template<typename... Args_lhs, typename... Args_rhs>
  bool **heterogeneous::operator**> (const heterovector< Args_lhs...> &lhs, const heterovector< Args_rhs...> &rhs)
- template<typename... Args_lhs, typename... Args_rhs>
  bool **heterogeneous::operator>=** (const heterovector< Args_lhs...> &lhs, const heterovector< Args_↩ rhs...> &rhs)

4.3.1 Detailed Description

This templated class replicates the features of an std::vector that can handle heterogeneous data types.