Gcombiner

A universal class library for combining objects from multiple containers.

HERA-B Note 01-002

See also HERA-B Note 01-001.

T. Glebe

Contents

1	**GC_trait** — ***Combiner***	3
2	**IteratorPair** — ***Combiner helper class***	5
3	**GCombiner** — ***Combiner***	7
4	**GCombinerIterator** — ***Iterates the combinations.***	9
5	**collision** — ***Collision function***	11
6	**SCombiner** — ***Combiner***	12
7	**SCombinerIterator** — ***Iterates the combinations.***	14
8	**GIteratorPair** — ***GSCombiner helper class***	16
9	**GSCombiner** — ***Combiner***	18
10	**GSCombinerIterator** — ***Iterates the combinations.***	20
11	**SDCombiner** — ***Combiner***	22
12	**SDCombinerIterator** — ***Iterates the combinations.***	24
13	**getComplement** — ***A Utility for SCombiner***	26
	Class Graph	27

1

template <class T> struct **GC_trait**

Combiner

Members

typedef T **Gvalue_type**

typedef T*
Gpointer

typedef T* const
Gconst_pointer

typedef T&
Greference

typedef const T&
Gconst_reference

typedef std::list <Gpointer>
GCombiner_v

typedef std::list <Gpointer> ::iterator
GCombiner_it

typedef std::list <Gpointer> ::const_iterator
GCombiner_cit

typedef std::list <Gpointer> ::reverse_iterator
GCombiner_rit

typedef std::list <Gpointer> ::const_reverse_iterator
GCombiner_rcit

typedef typename T::iterator
GIterator

typedef IteratorPair <GIterator>
ItPair

typedef std::list <ItPair>
GIterator_v

typedef std::list <ItPair> ::iterator
GIterator_it

typedef std::list <ItPair> ::const_iterator
GIterator_cit

typedef std::list <ItPair> ::reverse_iterator

GIterator_rit

typedef std::list <ItPair> ::const_reverse_iterator
GIterator_rcit

typedef std::vector <GIterator>
GCombination

typedef std::vector <GIterator> ::iterator
GCombination_it

typedef std::vector <GIterator> ::const_iterator
GCombination_cit

typedef std::vector <GIterator> ::reverse_iterator
GCombination_rit

typedef std::vector <GIterator> ::const_reverse_iterator
GCombination_rcit

Combiner trait. This trait class carries the definitions of containers and corresponding iterators which are used by the `GCombiner` and `GCombinerIterator` classes. For each kind of container `T` the internally used containers can be reconfigured by specializing this trait structure. The `GCombiner` class makes the following assumptions about the containers which should be combined:

1. all containers `T` must be of same type (e.g. vector<int>)
2. container has a `T::iterator` typedef (e.g. vector<int>::iterator)
3. container has a `begin()` method (e.g. vector<int>::begin())
4. container has an `end()` method (e.g. vector<int>::end())
5. `T::iterator` has an `operator*()` defined
6. `T::iterator` has an `operator==()` defined
7. `T::iterator` has an `operator!=()` defined
8. `T::iterator` has an `operator++()` defined

The above assumptions are fulfilled for all `STL` compatible containers.

Author: T. Glebe

2

template <class Iterator> class **IteratorPair**

Combiner helper class

Public Members

— Constructors —

IteratorPair ()

IteratorPair (const IteratorPair& rhs)

IteratorPair (const Iterator& a)

IteratorPair (const Iterator& lower, const Iterator& upper)

— Access methods —

inline bool
iteratable () const
has iter reached end?

inline self&
iterate () *do an iteration step*

— Operators —

inline bool
operator== (const IteratorPair<Iterator>& rhs) const

inline bool
operator!= (const IteratorPair<Iterator>& rhs) const

inline bool
operator< (const IteratorPair<Iterator>& rhs) const

— Expert functions —

inline void
reset ()

inline void
incr_reset ()

inline void

print (std::ostream& os) const

inline Iterator
getIter () const

GCombiner helper class. This class performs the bookkeeping of the begin, end and current iterator for each container saved by the Combiner class. The running interator can be resetted to the begin of the container in case it reached the end of the container. This class avoids that one has to declare the `GCombinerIterator` as a friend of the `GCombiner` class and thus reduces the coupling between those classes.

Note: This class is not for public use.

Author: T. Glebe

3 template <class T> class **GCombiner**

Combiner

Public Members

— Typedef's —

typedef GCombinerIterator <T>
iterator

typedef GCombinerIterator <T>
const_iterator

— Constructors —

GCombiner ()

GCombiner (const GCombiner<T>& rhs)

GCombiner (const Gpointer[],
const unsigned int N)

virtual ~**GCombiner** ()

— Access functions —

inline void
push_back (const Gvalue_type& rel)

inline Gpointer
back () const

inline int **size** () const

inline bool
empty () const

bool **valid** () const *check if combiner is valid: true if each container size is > 0*

— Iterator methods —

inline citerator
cbegin () const *to iterate container container, not available in CINT*

inline citerator
cend () const *to iterate container container, not available in CINT*

inline iterator

	begin () const	*to iterate combinations*
inline iterator	**end** () const	*to iterate combinations*
Gpointer	**operator[]** (unsigned int i) const	*direct access*
	— Operators —	
GCombiner <T> &	**operator=** (const GCombiner<T>& rhs)	
inline bool	**operator==** (const GCombiner<T>& rhs) const	
	— Expert functions —	
std::ostream&	**print** (std::ostream& os) const	
inline void	**clear** (void)	*empty combiner*
void	**deleteit** (void)	*empty combiner, call delete for pointers*
unsigned int	**ncomb** (void) const	*compute number of combinations*

Container class. The Combiner class stores pointers to containers whose elements should be combined. The number of containers is unlimited.

Author: T. Glebe

4

template <class T> class **GCombinerIterator**

Iterates the combinations.

Public Members

typedef GCombination_it
iterator

typedef GCombination_cit
const_iterator

— Constructors —

GCombinerIterator ()

GCombinerIterator (const GCombinerIterator<T>& rhs)

GCombinerIterator (const GCombiner_v& v, bool isend = false)

virtual ~**GCombinerIterator** ()

— Access methods —

inline GCombination_it
begin ()

inline GCombination_it
end ()

inline GCombination_cit
begin () const

inline GCombination_cit
end () const

inline int **size** () const

— Operators —

inline self&
operator++ ()

inline const GCombination&
operator* () const

inline const GIterator&

operator[] (unsigned int i) const

inline bool
operator== (const GCombinerIterator<T>& rhs) const

inline bool
operator!= (const GCombinerIterator<T>& rhs) const

— Expert functions —

std::ostream&
print (std::ostream& os) const

Protected Members

virtual void
iterate () *algorithm for an iteration step*

void **getCombination** () const

Combination iterator. The `GCombinerIterator` iterates over all combinations which are possible by the given container objects. The `GCombinerIterator` represents a vector of iterators of the container objects. Therefore this iterator behaves very different from a STL iterator and is not STL conform! The `GCombinerIterator` itself has a `begin()` and `end()` operator to loop over the elements of the current combination. An `operator*()` for receiving the vector of container iterators and an `operator[]()` method for direct access to the container iterators is provided.

Author: T. Glebe

5

template <class T> bool **collision** (GCombinerIterator<T>& rhs)

Collision function

Generic collision decection. In general, not every combination of objects from different containers is a useful one. In order to sort out those invalid combinations, one can define a collision function, which evaluates a current combination (represented by a `GCombinationIterator` if any two elements of the combination collides. In this case the function returns `true`, else `false`. This collision function detects a combination as invalid if any two objects are equal according to their `operator==()` method.

The user can define his own collision functions according to the function below and introduce other collision criterias. E.g. in case of containers carrying track objects one could detect a collision in case any track of the combination is a clone of another track in the combination.

In order to optimize this function, the containers must provide a `T::value_type` typedef which gives the storage type of the container. Example: `vector<int>::value_type` is `int`.

Author: T. Glebe

6

template <class T, unsigned int X> class **SCombiner**

Combiner

Public Members

— Typedef's —

typedef SCombinerIterator <T, X>
iterator *not available in CINT*

— Constructors —

SCombiner ()

SCombiner (Greference rhs)

— Iterator methods —

inline iterator
begin () const

inline iterator
end () const

— Access methods —

inline unsigned int
size () const *size of container*

inline void
set_container (Greference rhs)
set pointer to container

— Expert functions —

std::ostream&
print (std::ostream& os) const

inline unsigned long int
ncomb () const *no of combinations* $\frac{n!}{(n-k)!*k!}$

inline bool
valid () const *check if combiner is valid: true if size() >= X*

Combiner for a single container. SCombiner is a combiner which combines n objects from one container. The first template argument must be the type of a STL compatible container. n specifies the number of objects to combiner, it must be given as the second template argument. With SCombiner, each combination will occur only once. In a combination, the same object will not appear more than once.

It is assumed that the container `T` has the following properties:

1. a `T::begin()` and `T::end()` method
2. a `T::iterator` typedef
3. a `T::size()` method

Author: T. Glebe

7

template <class T, unsigned int X> class SCombinerIterator

Iterates the combinations.

Public Members

typedef GCombination_it **iterator**

typedef GCombination_cit **const_iterator**

— Constructors —

SCombinerIterator (Greference rhs, bool isend = false)

virtual ~**SCombinerIterator** ()

— Access methods —

inline GCombination_it **begin** ()

inline GCombination_it **end** ()

inline GCombination_cit **begin** () const

inline GCombination_cit **end** () const

inline int **size** () const

— Operators —

inline self& **operator++** ()

inline bool **operator==** (const self& rhs) const

inline bool **operator!=** (const self& rhs) const

inline const GCombination& **operator*** () const

inline const GIterator&

operator[] (unsigned int i) const

— Expert functions —

std::ostream&
print (std::ostream& os) const

Protected Members

virtual void
iterate () *algorithm for an iteration step*

Iterator for SCombiner. The `SCombinerIterator` iterates over all combinations which are possible by the given container object by excluding that the same object occurs more than once in a combination. The `SCombinerIterator` represents a vector of iterators pointing to the objects of the current combination. Therefore this iterator behaves very different from a STL iterator and is not STL conform! The `SCombinerIterator` itself has a `begin()` and `end()` method to loop over the elements of the current combination. An `operator*()` for receiving the vector of container iterators and an `operator[]()` method for direct access to the container iterators is provided.

Author: T. Glebe

8

template <class Iterator> class **GIteratorPair**

GSCombiner helper class

Public Members

— Constructors —

GIteratorPair ()

GIteratorPair (const GIteratorPair& rhs)

GIteratorPair (const Iterator& a)

GIteratorPair (const Iterator& lower,
const Iterator& upper)

— Access methods —

inline bool
iteratable () const
has iter reached end?

inline self&
iterate () *do an iteration step*

— Operators —

inline bool
operator== (const GIteratorPair<Iterator>& rhs)
const

inline bool
operator!= (const GIteratorPair<Iterator>& rhs)
const

inline bool
operator< (const GIteratorPair<Iterator>& rhs)
const

— Expert functions —

inline void
reset ()

inline std::ostream&
print (std::ostream& os) const

inline const Iterator&

getIter () const

GSCombiner helper class. This class performs the bookkeeping of the begin, end and current iterator for each container saved by the Combiner class. The running interator can be resetted to the begin of the container in case it reached the end of the container. This class avoids that one has to declare the `GSCombinerIterator` as a friend of the `GSCombiner` class and thus reduces the coupling between those classes.

Note: This class is not for public use.

Author: T. Glebe

9

template <class T> class **GSCombiner**

Combiner

Public Members

— Typedef's —

typedef GSCombinerIterator <T> **iterator**

typedef GSCombinerIterator <T> **const_iterator**

— Constructors —

GSCombiner ()

GSCombiner (const GSCombiner<T>& rhs)

GSCombiner (const Gpointer[], const unsigned int N)

~**GSCombiner** ()

— Access functions —

inline void **push_back** (const Gvalue_type& rel, unsigned int N)

void **push_back** (const Gvalue_type& rel)

inline int **size** () const — *no of containers used for combinatorics*

inline bool **empty** () const

bool **valid** () const — *check if combiner is valid: true if each container size is > 0*

— Iterator methods —

inline citerator **cbegin** () const — *to iterate container container, not available in CINT*

inline citerator

cend () const — *to iterate container container, not available in CINT*

inline iterator
begin () const — *to iterate combinations*

inline iterator
end () const — *to iterate combinations*

Gpointer **operator[]** (unsigned int i) const
direct access

— Operators —

GSCombiner <T> &
operator= (GSCombiner<T>& rhs)

inline bool
operator== (const GSCombiner<T>& rhs) const

— Expert functions —

std::ostream&
print (std::ostream& os) const

inline void
clear (void) — *empty combiner*

unsigned int
ncomb (void) const
compute number of combinations

Container class. The Combiner class stores pointers to containers whose elements should be combined. The number of containers is unlimited.

Author: T. Glebe

10

template <class T> class **GSCombinerIterator**

Iterates the combinations.

Public Members

typedef GCombination_it
iterator

typedef GCombination_cit
const_iterator

— Constructors —

GSCombinerIterator ()

GSCombinerIterator (const GSCombinerIterator<T>& rhs)

GSCombinerIterator (const GCombiner_v& v, bool isend = false)

virtual ~**GSCombinerIterator** ()

— Access methods —

inline GCombination_it
begin ()

inline GCombination_it
end ()

inline GCombination_cit
begin () const

inline GCombination_cit
end () const

inline unsigned int
size () const *no of elements in a combination*

— Operators —

inline self&
operator++ ()

inline const GCombination&

operator* () const
return vector with iterators of current combination

inline const typename GC_trait <T> ::GIterator&
operator[] (unsigned int i) const
access iterator i of current combination

inline bool
operator== (const GSCombinerIterator<T>& rhs) const

inline bool
operator!= (const GSCombinerIterator<T>& rhs) const

— Expert functions —

std::ostream&
print (std::ostream& os) const
used by operator<<()

Protected Members

virtual void
iterate () *algorithm for an iteration step*

void **getCombination** () const

Combination iterator. The `GCombinerIterator` iterates over all combinations which are possible by the given container objects. The `GCombinerIterator` represents a vector of iterators of the container objects. Therefore this iterator behaves very different from a STL iterator and is not STL conform! The `GCombinerIterator` itself has a `begin()` and `end()` operator to loop over the elements of the current combination. An `operator*()` for receiving the vector of container iterators and an `operator[]()` method for direct access to the container iterators is provided.

Author: T. Glebe

11 template <class T> class **SDCombiner**

Combiner

Public Members

— Typedef's —

typedef SDCombinerIterator <T>
iterator

— Constructors —

SDCombiner ()

SDCombiner (Greference rhs, unsigned int N)

— Iterator methods —

inline iterator
begin () const

inline iterator
end () const

— Access methods —

inline unsigned int
size () const *size of container*

inline void
set_container (Greference rhs)
set pointer to container

inline T* **pointer** () const *return pointer to Container*

inline unsigned int
N () const *combinatorics depth*

inline void
set_N (const unsigned int n)
set combinatorics depth

inline void
incr_N () *increment combinatorics depth*

— Expert functions —

std::ostream&
print (std::ostream& os) const

inline unsigned long int

ncomb () const *no of combinations* $\frac{n!}{(n-k)!*k!}$

Combiner for a single container. SDCombiner is a combiner which combines n objects from one container. The template argument must be the type of a STL compatible container. n specifies the number of objects to combiner, it must be given as a constructor argument. With SCombiner, each combination will occur only once. Within a combination, the same object will not appear more than once.

It is assumed that the container `T` has the following properties:

1. a `T::begin()` and `T::end()` method
2. a `T::iterator` typedef
3. a `T::size()` method

Author: T. Glebe

12

template <class T> class **SDCombinerIterator**

Iterates the combinations.

Public Members

typedef GCombination_it
iterator

typedef GCombination_cit
const_iterator

— Constructors —

SDCombinerIterator (Greference rhs,
unsigned int X,
bool isend = false)

virtual ~**SDCombinerIterator** ()

— Access methods —

inline GCombination_it
begin ()

inline GCombination_it
end ()

inline GCombination_cit
begin () const

inline GCombination_cit
end () const

inline int **size** () const

— Operators —

inline self&
operator++ ()

inline const GCombination&
operator* () const

inline const GIterator&
operator[] (unsigned int i) const

inline bool

operator== (const SDCombinerIterator<T>& rhs) const

inline bool
operator!= (const SDCombinerIterator<T>& rhs) const

— Expert functions —

std::ostream&
print (std::ostream& os) const

Protected Members

virtual void
iterate () *algorithm for an iteration step*

Iterator for SDCombiner. The `SDCombinerIterator` iterates over all combinations which are possible by the given container object by excluding that the same object occurs more than once in a combination. The `SCombinerIterator` represents a vector of iterators pointing to the objects of the current combination. Therefore this iterator behaves very different from a STL iterator and is not STL conform! The `SCombinerIterator` itself has a `begin()` and `end()` method to loop over the elements of the current combination. An `operator*()` for receiving the vector of container iterators and an `operator[]()` method for direct access to the container iterators is provided.

Author: T. Glebe

13

template <class T, unsigned int X> bool **getComplement** (const SCombinerIterator<T, X>& sc, T& input, T& output)

A Utility for SCombiner

A Utility for SCombiner. getComplement() stores all objects from container `T` which are not present in the given SCombinerIterator in the container given as third argument. Thus the complement with respect to the objects in the SCombinerIterator can be accessed.

The container which is searched for matches with objects from the SCombinerIterator and the return container have to be of the same type as the one used by SCombiner to build the combinations.

Author: T. Glebe, Michaela Agari

Class Graph

2 IteratorPair 5

3 GCombiner 7

4 GCombinerIterator 9

6 SCombiner 12

7 SCombinerIterator 14

8 GIteratorPair 16

9 GSCombiner	....	18
10 GSCombinerIterator	....	20
11 SDCombiner	....	22
12 SDCombinerIterator	....	24