Concept reference

Concepts

Concept Assignable

Assignable

Description

Assignable types must have copy constructors, `operator=` for assignment, and the `swap()` function defined.

Refinement of

- CopyConstructible

Notation

X	A type playing the role of assignable-type in the Assignable concept.
x, y	Objects of type X

Valid expressions

Name	Expression	Type	Semantics
Assignment	x = y	X &	Require `operator=`
Swap	swap(x, y)	void	Require `swap()` function

Models

- int

See also

- CopyConstructible

Concept InputIterator

InputIterator

Description

An input iterator is an iterator that can read through a sequence of values. It is single-pass (old values of the iterator cannot be re-used), and read-only.

An input iterator represents a position in a sequence. Therefore, the iterator can point into the sequence (returning a value when dereferenced and being incrementable), or be off-the-end (and not dereferenceable or incrementable).

Refinement of

- Assignable
- DefaultConstructible
- EqualityComparable

Associated types

- **value_type**

  ```
  std::iterator_traits<Iter>::value_type
  ```

  The value type of the iterator (not necessarily what `*i` returns)

- **difference_type**

  ```
  std::iterator_traits<Iter>::difference_type
  ```

  The difference type of the iterator

- **category**

  ```
  std::iterator_traits<Iter>::iterator_category
  ```

  The category of the iterator

Notation

Iter	A type playing the role of iterator-type in the InputIterator concept.
`i`, `j`	Objects of type Iter
`x`	Object of type value_type

Type expressions

Category tag	category must be derived from std::input_iterator_tag, a model of DefaultConstructible, and a model of CopyConstructible.
Value type copy constructibility	value_type must be a model of CopyConstructible.
Difference type properties	difference_type must be a model of SignedInteger.

Valid expressions

Name	Expression	Type	Precondition	Semantics	Postcondition
Dereference	*i	Convertible to value_type	`i` is incrementable (not off-the-end)		
Preincrement	++i	Iter &	`i` is incrementable (not off-the-end)		
Postincrement	i++		`i` is incrementable (not off-the-end)	Equivalent to `(void)(++i)`	`i` is dereference-able or off-the-end
Postincrement and dereference	*i++	Convertible to value_type	`i` is incrementable (not off-the-end)	Equivalent to `{value_type t = *i; ++i; return t;}`	`i` is dereference-able or off-the-end

Complexity

All iterator operations must take amortized constant time.

Models

- std::istream_iterator

See also

- DefaultConstructible
- EqualityComparable
- ForwardIterator
- OutputIterator

Concept OutputIterator

OutputIterator

Description

An output iterator is an iterator that can write a sequence of values. It is single-pass (old values of the iterator cannot be re-used), and write-only.

An output iterator represents a position in a (possibly infinite) sequence. Therefore, the iterator can point into the sequence (returning a value when dereferenced and being incrementable), or be off-the-end (and not dereferenceable or incrementable).

Associated types

- **value_type**

  ```
  std::iterator_traits<Iter>::value_type
  ```

  The stated value type of the iterator (should be `void` for an output iterator that does not model some other iterator concept).

- **difference_type**

  ```
  std::iterator_traits<Iter>::difference_type
  ```

  The difference type of the iterator

- **category**

  ```
  std::iterator_traits<Iter>::iterator_category
  ```

  The category of the iterator

Notation

Iter	A type playing the role of iterator-type in the OutputIterator concept.
ValueType	A type playing the role of value-type in the OutputIterator concept.
`i`, `j`	Objects of type Iter
`x`	Object of type ValueType

Type expressions

	The type Iter must be a model of Assignable.
	The type ValueType must be a model of Assignable.
	The type Iter must be a model of DefaultConstructible.
	The type Iter must be a model of EqualityComparable.
Category tag	category must be derived from std::output_iterator_tag, a model of DefaultConstructible, and a model of CopyConstructible.
Difference type properties	difference_type must be a model of SignedInteger.

Valid expressions

Name	Expression	Type	Precondition	Semantics	Postcondition
Dereference	*i		`i` is incrementable (not off-the-end)		
Dereference and assign	*i = x		`i` is incrementable (not off-the-end)		`*i` may not be written to again until it has been incremented.
Preincrement	++i	Iter &	`i` is incrementable (not off-the-end)		
Postincrement	i++		`i` is incrementable (not off-the-end)	Equivalent to `(void)(++i)`	`i` is dereferenceable or off-the-end
Postincrement, dereference, and assign	*i++ = x		`i` is incrementable (not off-the-end)	Equivalent to `{*i = t; ++i;}`	`i` is dereferenceable or off-the-end

Complexity

All iterator operations must take amortized constant time.

Models

- std::ostream_iterator, ...
- std::insert_iterator, ...
- std::front_insert_iterator, ...
- std::back_insert_iterator, ...

Concept ForwardIterator

ForwardIterator

Description

A forward iterator is an iterator that can read through a sequence of values. It is multi-pass (old values of the iterator can be re-used), and can be either mutable (data pointed to by it can be changed) or not mutable.

An iterator represents a position in a sequence. Therefore, the iterator can point into the sequence (returning a value when dereferenced and being incrementable), or be off-the-end (and not dereferenceable or incrementable).

Refinement of

- InputIterator
- OutputIterator

Associated types

- **value_type**

  ```
  std::iterator_traits<Iter>::value_type
  ```

  The value type of the iterator

- **category**

  ```
  std::iterator_traits<Iter>::iterator_category
  ```

  The category of the iterator

Notation

Iter	A type playing the role of iterator-type in the ForwardIterator concept.
`i`, `j`	Objects of type Iter
`x`	Object of type value_type

Type expressions

Category tag	category must be derived from std::forward_iterator_tag.

Valid expressions

Name	Expression	Type	Precondition	Semantics	Postcondition
Dereference	*i	const-if-not-mutable value_type &	`i` is incrementable (not off-the-end)		
Member access	i->{member-name} (return type is pointer-to-object type)	const-if-not-mutable value_type *	`i` is incrementable (not off-the-end)		
Preincrement	++i	Iter &	`i` is incrementable (not off-the-end)		
Postincrement	i++	Iter	`i` is incrementable (not off-the-end)	Equivalent to `{Iter j = i; ++i; return j;}`	`i` is dereference-able or off-the-end

Complexity

All iterator operations must take amortized constant time.

Invariants

Predecrement must return object	`&i = &(++i)`
Unique path through sequence	`i == j` implies `++i == ++j`

Models

- T *
- std::hash_set<T>::iterator

See also

- BidirectionalIterator

Concept BidirectionalIterator

BidirectionalIterator

Description

A bidirectional iterator is an iterator that can read through a sequence of values. It can move in either direction through the sequence, and can be either mutable (data pointed to by it can be changed) or not mutable.

An iterator represents a position in a sequence. Therefore, the iterator can point into the sequence (returning a value when dereferenced and being incrementable), or be off-the-end (and not dereferenceable or incrementable).

Refinement of

- ForwardIterator

Associated types

- **value_type**

```
std::iterator_traits<Iter>::value_type
```

  The value type of the iterator

- **category**

```
std::iterator_traits<Iter>::iterator_category
```

  The category of the iterator

Notation

Iter	A type playing the role of iterator-type in the BidirectionalIterator concept.
`i`, `j`	Objects of type Iter
`x`	Object of type value_type

Type expressions

Category tag	category must be derived from std::bidirectional_iterator_tag.

Valid expressions

Name	Expression	Type	Precondition	Semantics	Postcondition
Predecrement	--i	Iter &	`i` is incrementable (not off-the-end) and some dereferenceable iterator `j` exists such that `i == ++j`		
Postdecrement	i--	Iter	Same as for predecrement	Equivalent to `{Iter j = i; --i; return j;}`	`i` is dereferenceable or off-the-end

Complexity

All iterator operations must take amortized constant time.

Invariants

Predecrement must return object	`&i = &(--i)`
Unique path through sequence	`i == j` implies `--i == --j`
Increment and decrement are inverses	`++i; --i;` and `--i; ++i;` must end up with the value of `i` unmodified, if `i` both of the operations in the pair are valid.

Models

- T *
- std::list<T>::iterator

See also

- RandomAccessIterator

Concept RandomAccessIterator

RandomAccessIterator

Description

A random access iterator is an iterator that can read through a sequence of values. It can move in either direction through the sequence (by any amount in constant time), and can be either mutable (data pointed to by it can be changed) or not mutable.

An iterator represents a position in a sequence. Therefore, the iterator can point into the sequence (returning a value when dereferenced and being incrementable), or be off-the-end (and not dereferenceable or incrementable).

Refinement of

- BidirectionalIterator
- LessThanComparable

Associated types

- **value_type**

  ```
  std::iterator_traits<Iter>::value_type
  ```

  The value type of the iterator

- **category**

  ```
  std::iterator_traits<Iter>::iterator_category
  ```

  The category of the iterator

- **difference_type**

  ```
  std::iterator_traits<Iter>::difference_type
  ```

  The difference type of the iterator (measure of the number of steps between two iterators)

Notation

Iter	A type playing the role of iterator-type in the RandomAccessIterator concept.
`i`, `j`	Objects of type Iter
`x`	Object of type value_type
`n`	Object of type difference_type
`int_off`	Object of type int

Type expressions

Category tag	category must be derived from std::random_access_iterator_tag.

Valid expressions

Name	Expression	Type	Semantics
Motion	i += n	Iter &	Equivalent to applying `i++` `n` times if `n` is positive, applying `i-- -n` times if `n` is negative, and to a null operation if `n` is zero.
Motion (with integer offset)	i += int_off	Iter &	Equivalent to applying `i++` `n` times if `n` is positive, applying `i-- -n` times if `n` is negative, and to a null operation if `n` is zero.
Subtractive motion	i -= n	Iter &	Equivalent to `i+=(-n)`
Subtractive motion (with integer offset)	i -= int_off	Iter &	Equivalent to `i+=(-n)`
Addition	i + n	Iter	Equivalent to `{Iter j = i; j += n; return j;}`
Addition with integer	i + int_off	Iter	Equivalent to `{Iter j = i; j += n; return j;}`
Addition (count first)	n + i	Iter	Equivalent to `i + n`
Addition with integer (count first)	int_off + i	Iter	Equivalent to `i + n`
Subtraction	i - n	Iter	Equivalent to `i + (-n)`
Subtraction with integer	i - int_off	Iter	Equivalent to `i + (-n)`
Distance	i - j	difference_type	The number of times `i` must be incremented (or decremented if the result is negative) to reach `j`. Not defined if `j` is not reachable from `i`.
Element access	i[n]	const-if-not-mutable value_type &	Equivalent to `*(i + n)`
Element access with integer index	i[int_off]	const-if-not-mutable value_type &	Equivalent to `*(i + n)`

Complexity

All iterator operations must take amortized constant time.

Models

- T *
- std::vector<T>::iterator

- std::vector<T>::const_iterator
- std::deque<T>::iterator
- std::deque<T>::const_iterator

See also

- LessThanComparable

Concept DefaultConstructible

DefaultConstructible

Description

DefaultConstructible objects only need to have a default constructor.

Notation

X   A type playing the role of default-constructible-type in the DefaultConstructible concept.

Valid expressions

Name	Expression	Type	Semantics
Construction	X()	X	Construct an instance of the type with default parameters.

Models

- int
- std::vector<double>

Concept CopyConstructible

CopyConstructible

Description

Copy constructible types must be able to be constructed from another member of the type.

Notation

X	A type playing the role of copy-constructible-type in the CopyConstructible concept.
x, y	Objects of type X

Valid expressions

Name	Expression	Type	Semantics
Copy construction	X(x)	X	Require copy constructor.

Models

- int

Concept EqualityComparable

EqualityComparable

Description

Equality Comparable types must have == and != operators.

Notation

X	A type playing the role of comparable-type in the EqualityComparable concept.
x, y	Objects of type X

Valid expressions

Name	Expression	Type
Equality test	x == y	Convertible to bool
Inequality test	x != y	Convertible to bool

Models

- int
- std::vector<int>

Concept LessThanComparable

LessThanComparable

Description

LessThanComparable types must have <, >, <=, and >= operators.

Notation

X	A type playing the role of comparable-type in the LessThanComparable concept.
x, y	Objects of type X

Valid expressions

Name	**Expression**	**Type**	**Semantics**
Less than	x < y	Convertible to bool	Determine if one value is less than another.
Less than or equal	x <= y	Convertible to bool	Determine if one value is less than or equal to another.
Greater than	x > y	Convertible to bool	Determine if one value is greater than another.
Greater than or equal to	x >= y	Convertible to bool	Determine if one value is greater than or equal to another.

Models

- int

Concept SignedInteger

SignedInteger

Refinement of

- CopyConstructible
- Assignable
- DefaultConstructible
- EqualityComparable
- LessThanComparable

Notation

T	A type playing the role of integral-type in the SignedInteger concept.
x, y, z	Objects of type T
a, b	Objects of type int

Type expressions

Conversion to int	T must be convertible to int.

Valid expressions

19

Name	Expression	Type
Conversion from int	T(a)	T
Preincrement	++x	T &
Predecrement	--x	T &
Postincrement	x++	T
Postdecrement	x--	T
Sum	x + y	T
Sum with int	x + a	T
Sum-assignment	x += y	T &
Sum-assignment with int	x += a	T &
Difference	x - y	T
Difference with int	x - a	T
Product	x * y	T
Product with int	x * a	T
Product-assignment with int	x *= a	T &
Product with int on left	a * x	T
Quotient	x / y	T
Quotient with int	x / a	T
Right-shift	x >> y	T
Right-shift with int	x >> a	T
Right-shift-assignment with int	x >>= a	T &
Less-than comparison	x < y	Convertible to bool
Less-than comparison with int	x < a	Convertible to bool
Less-than comparison with size_t	x < boost::sample_value < std::size_t >()	Convertible to bool
Greater-than comparison	x > y	Convertible to bool
Greater-than comparison with int	x > a	Convertible to bool
Less-than-or-equal comparison	x <= y	Convertible to bool
Less-than-or-equal comparison with int	x <= a	Convertible to bool
Greater-than-or-equal comparison	x >= y	Convertible to bool

Name	**Expression**	**Type**
Greater-than-or-equal comparison with int	x >= a	Convertible to bool
Greater-than-or-equal comparison with int on left	a >= x	Convertible to bool
Equality comparison	x == y	Convertible to bool
Equality comparison with int	x == a	Convertible to bool