Doc No: X3J16/95-0100
WG21/N0700
Date: May 30, 1995
Project: Programming Language C++
Reply to: Beman Dawes
beman@dawes.win.net

Requirements for STL Template Arguments

The working paper specifies in 17.3.3.6 [lib.res.on.functions] that types used as template arguments to instantiate library components must meet the appropriate requirements. Section 17.2.1.2 [lib.structure.requirements] says that "Requirements are stated in terms of well-defined expressions..."

This is the approach taken by Alex Stepanov and Meng Lee in specifying requirements for template arguments for STL allocators [lib.allocator.requirements], containers [lib.container.requirements, etc.] and iterators [lib.iterator.requirements, etc.].

The purpose of this proposal is to specify requirements for various other library template arguments for which the working paper does not currently specify requirements. This will resolve a LWG open issue.

The first part of the proposal specifies several requirements on types "in terms of well-defined expressions." Each requirement is given a name for ease of reference. A requirement name does not identify a specific type but rather identifies all types (built-in or programmer defined) which meet the specified requirements.

As always, any names introduced in this proposal serve as placeholders for final names to be chosen by the LWG.

The second part of the proposal specifies which requirements apply to various library template arguments.

Some readers may find it easier to skim read Part 2 to get a feel for usage before reading the detailed requirements in Part 1.

See lib-3714 through lib-3718 for examples of questions this proposal attempts to resolve.

Proposal Part 1 - Requirements Specification

The form and content of these requirements were chosen to be very similar to the STL iterator and container requirements currently in the WP.

(Comment: In lib-3754 John Max Skaller questions the definitions of some of the terms used in these requirements, such as "equivalent" and "equivalence relationship". For purposes of this proposal such terms are understood to have the same meaning as they do in the Iterators and Containers chapters of the WP.)

CopyConstructible Requirements

T is the type, t is a value of T, u is a value of const T.

Expression	Return Type	Assertions/etc.	Complexity
T(t)		post: t is equivalent to T(t)	constant
T(u)		post: u is equivalent to T(u)	constant
t.~T();			constant
&t	T*	post: denotes the address of t	constant
&u	const T*	post: denotes the address of u	constant

The default constructor is not required. Certain container class member function signatures specify the default constructor as a default argument. T() must be a well-defined expression if one of those signatures is called using the default argument.

Assignable Requirements

Assignable types must also meet requirements for CopyConstructible types.

T is the type, t is a value of T, u is a value of (possibly const) T.

Expression	Return Type	Assertions/etc.	Complexity
t = u	T&	post: t is equivalent to u	constant

(Comment: The point in separating CopyConstructible and Assignable is that CopyConstructibles can be const, but Assignables can’t be const.)

EqualityComparable Requirements

T is the type, a and b are values of T.

Expression	Return Type	Assertions/etc.	Complexity
a == b	convertible to bool	== is an equivalence relationship	constant

LessThanComparable Requirements

T is the type, a and b are values of type T.

Expression	Return Type	Assertions/etc.	Complexity
a < b	convertible to bool	< is a total ordering relationship	constant

(Comment: In lib-3754 John Max Skaller indicates his belief that this requirement needs modification, both in this proposal and where it is already used for STL components in the WP. He suggests:

> "Let S be the set of all expressions of the form *I where i is in the range specified in the input of the algorithm, THEN for each pair of expressions *i1 and *i2 in S, *i1 < *i2 is well defined, functional, and convertible to bool, and is a total order on S."

This proposal does not include that wording. I see it as incomplete (in the case of many algorithms which take a value argument and/or multiple range arguments) and already implied by the simpler wording currently in the WP. If someone wants to make another proposal clarifying the wording, that's fine, but that is not part of this proposal.)

Comparable Requirements

EqualityComparable and LessThanComparable.

Proposal Part 2 - Requirements for various template arguments

These requirements are additions to any template argument requirements already specified in the working paper.

Requirements on template arguments:

Template	Argument	Requirements	Reference
`operator!=`	`T`	EqualityComparable	lib.operators
`operator>`	`T`	LessThanComparable	lib.operators
`operator<=`	`T`	LessThanComparable	lib.operators
`operator>=`	`T`	LessThanComparable	lib.operators
`pair`	`T1, T2`	CopyConstructible	lib.pairs
`plus`	`T`	`t + t` returns T	lib.arithmetic.operations
`minus`	`T`	`t - t` returns T	lib.arithmetic.operations
`times`	`T`	`t * t` returns T	lib.arithmetic.operations
`divides`	`T`	`t / t` returns T	lib.arithmetic.operations
`modulus`	`T`	`t % t` returns T	lib.arithmetic.operations
`negate`	`T`	`- t` returns T	lib.arithmetic.operations
`equal_to`	`T`	`t == t` return convertible to `bool`	lib.comparisons
`not_equal_to`	`T`	`t != t` return convertible to `bool`	lib.comparisons
`greater`	`T`	`t > t` return convertible to `bool`	lib.comparisons
`less`	`T`	`t < t` return convertible to `bool`	lib.comparisons
`greater_equal`	`T`	`t >= t` return convertible to `bool`	lib.comparisons
`less_equal`	`T`	`t <= t` return convertible to `bool`	lib.comparisons
`logical_and`	`T`	`t && t` return convertible to `bool`	lib.logical.operations
`logical_or`	`T`	`t || t` return convertible to `bool`	lib.logical.operations
`logical_no`	`T`	`!t` return convertible to `bool`	lib.logical.operations
`deque`	`T`	Assignable	lib.deque
`list`	`T`	Assignable	lib.list
`vector`	`T`	Assignable	lib.vector
`map`	`Key`	Assignable	lib.map
`map`	`T`	Assignable	lib.map
`multimap`	`Key`	Assignable	lib.multimap
`multimap`	`T`	Assignable	lib.multimap
`set`	`Key`	Assignable	lib.set
`multiset`	`Key`	Assignable	lib.multiset
`find`	`T`	EqualityComparable	lib.alg.find
`count`	`T`	EqualityComparable	lib.alg.count

search	T	EqualityComparable	lib.alg.search
swap	T	Assignable	lib.alg.swap
replace	T	EqualityComparable, Assignable	lib.alg.replace
replace_if	T	EqualityComparable, Assignable	lib.alg.replace
replace_copy	T	EqualityComparable, Assignable	lib.alg.replace
replace_ copy_if	T	EqualityComparable, Assignable	lib.alg.replace
fill	T	EqualityComparable, Assignable	lib.alg.fill
fill_n	T	EqualityComparable, Assignable	lib.alg.fill
remove	T	EqualityComparable	lib.alg.remove
remove_ copy	T	EqualityComparable	lib.alg.remove
lower_bound	T	LessThanComparable	lib.lower.bound
upper_bound	T	LessThanComparable	lib.upper.bound
equal_range	T	LessThanComparable	lib.equal.range
binary_search	T	LessThanComparable	lib.binary.search
min	T	LessThanComparable, CopyConstructible	lib.min
max	T	LessThanComparable, CopyConstructible	lib.max
count	Size	operator++()	lib.alg.count
count_if	Size	operator++()	lib.alg.count
search	Size	convertible to integral type	lib.alg.search
fill_n	Size	convertible to integral type	lib.alg.fill
generate_n	Size	convertible to integral type	lib.alg.generate
all uses	Predicate	As specified in reference	lib.algorithms
all uses	BinaryPredicate	As specified in reference	lib.algorithms
all uses	Allocator	As specified in reference	lib.allocator.requirements
all uses	InputIterator	As specified in reference	lib.input.iterators
all uses	OutputIterator	As specified in reference	lib.output.iterators
all uses	ForwardIterator	As specified in reference	lib.forward.iterators
all uses	BidirectionalIterator	As specified in reference	lib.bidirectional.iterators
all uses	RandomAccessIterator	As specified in reference	lib.random.access.iterators
all uses	Container	As specified in reference	lib.container.requirements
all uses	Compare	As specified in reference	??????
all uses	Function	As specified in reference	lib.function.objects

(Comment: For many STL algorithms T is not required by this proposal to be Assignable. This may overconstrain implementations in that it may disallow some optimizations such as value caching.

(Comment: There are still some template arguments with unspecified requirements, but I have run out of time and energy)