Generalized Pointer Types for Allocators J16/97-0009 = WG21/N1047

Greg Colvin. Information Management Research.

I propose here a set of requirements for generalized pointer types. These are not the only possible requirements, but I believe them to be reasonable, consistent with existing practice, implementable by allocators, and usable by containers.

Changes to 20.1.5 [lib.allocator.requirements]

In the table "Descriptive variable definitions", change the definition of variable *u* and define a variable *v*, as follows:

u	a value of type **Y::pointer** obtained by calling **Y::allocate**, or else **0**.
v	a value of type **Y::const_pointer** obtained by conversion from a value **u**.

In the table "Allocator requirements", replace the reference to *u* with *v*, change the requirements for *construct* and *destroy*, and add requirements for assignment and copy construction as follows:

a.allocate(n,v)	...	...
a.construct(p,t)	(not used)	Post: ***p == T(t)**
a.destroy(p)	(not used)	Effect: **p->~T()**
u = p	**Y::pointer**	Pre: **T*** can be implicitly converted to **U***
		Post: **u == p**
v = p	**Y::const_pointer**	Pre: **T*** can be implicitly converted to **U***
		Post: **v == p**
v = q	**Y::const_pointer**	Pre: **T*** can be implicitly converted to **U***
		Post: **v == q**
Y::pointer w(p);		Pre: **T*** can be implicitly converted to **U***
		Post: **w == p**
Y::const_pointer x(p);		Pre: **T*** can be implicitly converted to **U***
		Post: **x == p**
Y::const_pointer x(q);		Pre: **T*** can be implicitly converted to **U***
		Post: **x == q**

Replace paragraphs 4 and 5 with the following sentence:

> The semantics of containers and algorithms when allocator instances compare non-equal, are implementation defined.

I would much prefer the stronger requirement, and existing practice, that the complexity of operations involving non-equal allocator instances be linear, but that requirement is controversial, and would require changes to every *swap* and *splice* operation in the draft.

Changes to 21.3.1 [lib.string.cons]

Append the following sentence to paragraph 1:

> The **Allocator** argument must meet the further requirement that the typedef members **pointer**, **const_pointer**, **size_type**, and **difference_type** be **charT***, **const charT***, **size_t**, and **ptrdiff_t**, respectively.

Changes not made

The draft, in **20.1.5**, requires the result of *Allocator::allocate* to be a random access iterator; I see no need for further requirements on *Allocator::pointer* semantics. If such need is demonstrated then I expect it will be iterators that require tighter specification. Neither have I attempted to generalize reference types, given the impossiblity of defining a member access operator, and given the definition in **24.1.1** of iterator member access in terms of reference member access.