

Doc No: N2666
Reply to: Matt Austern <austern@google.com>
Date: 2008-06-11=08-0176

More STL algorithms (revision 2)

This paper proposes a number of nonstandard STL-style algorithms for inclusion in the standard. Nothing in this paper is novel or complicated. All of these algorithms have been around for years, and have been described in books and provided by library vendors as nonstandard extensions. All of them are easily implementable, and have been implemented. Most of them are taken from the SGI STL (<http://www.sgi.com/tech/stl>).

This proposal is a pure extension.

This paper has been updated to reflect Library Working Group comments from the June 2008 Sophia Antipolis standards committee meeting. Some algorithms have been removed, some have been renamed, and there have been minor changes in wording.

Design Decisions

In most cases there aren't any very important decisions to be made once we've decided to provide the algorithm in the first place.

Counting versions of copy

The standard `copy` algorithm takes three arguments: a first/last input iterator pair to determine the input range, and a single output iterator to determine the start of the output range. Sometimes, however, it's more convenient to describe the operation as a starting point and a count than as a starting point and an ending point. This is usually just a matter of convenience, since we can usually convert between those forms by writing `n = distance(first, last)` or `last = first, advance(last, n)`, but it can occasionally be more important if the iterators are of a type where converting between a range and a count would be expensive or impractical. (Input iterators that aren't forward iterators, for example.) The SGI STL provides `copy_n` and the corresponding version for initializing raw memory, `uninitialized_copy_n`.

all_of, any_of, and none_of

These three algorithms provide the ordinary mathematical operations \forall , \exists , and $\#$: given a range and a predicate, determine whether that predicate is true for all elements; whether there exists an element for which the predicate is true; or whether there exist no elements for which the predicate is true. Strictly speaking there is no need to provide all three of these algorithms (`!none_of` and `any_of` are equivalent), but all three of these operations feel equally fundamental.

These three algorithms have accumulated various names over the years. The original version of this paper chose the names `any/none/all` (taken from the latest draft of Alex Stepanov and Paul McJones's *Elements of Programming*). At the request of the LWG, this revision uses the names `all_of/any_of/none_of`.

