Doc. no.: P0808R0
Date: 2017-10-12
Reply to: Titus Winters (titus@google.com)
Audience: LEWG

Ranges Naming

Abstract

As the Ranges TS progresses toward adoption into the main standard it is time to examine the naming choices in that library for consistency with the rest of the standard. This paper aims to point out one major area for concern in naming - using a “view” to modify underlying storage, directly contradicting the use of “view” in names like string_view - and suggest possible approaches to rename pieces of the Ranges TS.

Range Concepts

The Ranges TS specifies three primary Concepts around Ranges in [ranges.requirements.general] - Range, SizedRange, and View.

- “The Range concept requires only that begin and end return an iterator and a sentinel.”
- “The SizedRange concept refines Range with the requirement that the number of elements in the range can be determined in constant time using the size function.”
- “The View concept specifies requirements on an (sic) Range type with constant-time copy and assign operations.”

Anecdotally, View is following from the naming precedent set by SQL in which a View is a logical table, but doesn’t own any of its data - it merely defines windows into the underlying tables and synthesizes a logical table from those. However, the English word “view” has strong connotations of immutability: you cannot use a view to modify that which is being viewed. (Consider: viewpoint, viewing area, ViewMaster.) Further, C++ has already utilized the “view is immutable” terminology in published standards: C++17 includes std::string_view. A string_view has constant-time copy and assignment (logically it is a pointer + length), but additionally promises that the underlying data cannot be modified via the view.

This tension is a primary motivator for this paper - before the Ranges TS is adopted into the standard we want to make an effort to resolve this, either by finding a currently-unused term for cheap-to-copy (often lazily evaluated) ranges that does not imply immutability, or by making up a new term if necessary.

Alternative names for “View”

- Span - already basically taken for the std::span proposals - the general form of mutable (ptr + length). However, we can envision SpanRange or the like as the Range/Concept name for a contiguous range.
- ~~Generator - some precedent in Python, nothing in C++, maybe OK although it sorta means a different thing (and would get in the way of some coroutines options, potentially)¶~~
- Stream - some precedent in Javascript, although JS Streams are asynchronous
- Reach - more commonly a verb, but that could be overcome (no software precedent I’m aware of)
- Spread - odd connotation, also more commonly a verb
- Extent - already exists, the std::extent metafunction
- Interval - maybe, but that precludes any std::interval concept for intervals on a number line.
- LazyRange - Editorial descriptions in the Ranges TS describe views as “composable, non-mutating, lazy algorithms over ranges” - we could follow in the footsteps of iterators and use one core concept (Ranges) and add adjectives as necessary to refine.
    - O1Range
    - ReferenceRange - only covers a subset of Views, but maybe an important subset
    - ConstantRange - again only a subset
    - ContiguousRange - More for spans than Views
- Make up a completely new term

I propose that LEWG spend some time discussing at least these names (with the input of the Ranges TS experts, of course). Personally, I like LazyRange as of this writing.

Rejected names for "View"

- Generator - existing practice in Python and around some coroutines options defines a generator as a function, which is a way to create an iterator or range, but isn't one itself.

Range Namespaces

I reiterate the position that I put forward on the lib-ext reflector during the Toronto meeting: we should stop adding nested namespaces in the main standard. (I don’t mind for the purposes of TSes.) Although there has been no proposal as of yet to merge the Ranges TS into the working paper, I would like to seek consensus that Range naming is handled in such a way to avoid the introduction of a std::ranges sub-namespace, or future std2:: subnamespaces such as std2::view.