Document number: N4280
Date: 2014-11-06
Project: Programming Language C++, Library Working Group
Reply-to: Riccardo Marcangelo <ricky.65@outlook.com>

Non-member size() and more (Revision 2)

Introduction

This paper revises N4155 "Non-member size() and more (Revision 1)" in response to feedback from the LWG. Please see the original paper, N4017, for the rationale behind this proposal.

This paper includes the following change from N4155 as requested by the LWG:

- The proposed wording for overloads taking arbitrary C&/const C& now use a *trailing-return-type*.
- empty() for initializer_list is now specified with "Returns".

Proposed Wording

Modify the section 24.3 Header <iterator> synopsis [iterator.synopsis] by adding the following at the end of the synopsis:

// 24.8, container access:

template <class C> constexpr auto size(const C& c) -> decltype(c.size());

template <class T, size_t N> constexpr size_t size(const T (&array)[N]) noexcept;

template <class C> constexpr auto empty(const C& c) -> decltype(c.empty());

template <class T, size_t N> constexpr bool empty(const T (&array)[N]) noexcept;

template <class E> constexpr bool empty(initializer_list<E> il) noexcept;

template <class C> constexpr auto data(C& c) -> decltype(c.data());

template <class C> constexpr auto data(const C& c) -> decltype(c.data());

template <class T, size_t N> constexpr T* data(T (&array)[N]) noexcept;

template <class E> constexpr const E* data(initializer_list<E> il) noexcept;

Add a new section 24.8 Container access [iterator.container] containing the following:

In addition to being available via inclusion of the <iterator> header, the function templates in [iterator.container] are available when any of the following headers are included: <array>, <deque>, <forward_list>, <list>, <map>, <regex>, <set>, <string>, <unordered_map>, <unordered_set>, and <vector>.

template <class C> constexpr auto size(const C& c) -> decltype(c.size());

Returns: c.size().

template <class T, size_t N> constexpr size_t size(const T (&array)[N]) noexcept;

Returns: N.

template <class C> constexpr auto empty(const C& c) -> decltype(c.empty());

Returns: c.empty().

template <class T, size_t N> constexpr bool empty(const T (&array)[N]) noexcept;

Returns: false.

template <class E> constexpr bool empty(initializer_list<E> il) noexcept;

Returns: il.size() == 0.

template <class C> constexpr auto data(C& c) -> decltype(c.data());

template <class C> constexpr auto data(const C& c) -> decltype(c.data());

Returns: c.data().

template <class T, size_t N> constexpr T* data(T (&array)[N]) noexcept;

Returns: array.

template <class E> constexpr const E* data(initializer_list<E> il) noexcept;

Returns: il.begin().

Acknowledgments

A big thank you to Stephan T. Lavavej for his numerous and valuable feedback, suggestions, and corrections for this proposal.

Credit to Daniel Krügler for suggesting "container access" as the name of the new sub-clause in <iterator>.