Proposal to the ISO C++ Standards Committee: Introduction of `std::contains`, `std::contains_any`, and `std::contains_all`

Robert Sitton

February 29, 2024

Abstract

This proposal introduces `std::contains`, `std::contains_any`, and `std::contains_all` to the C++ Standard Library, enabling direct, efficient queries on containers and ranges.

1 Rationale

Current C++ lacks straightforward mechanisms for querying container contents. Developers resort to verbose, error-prone patterns. This proposal addresses this gap, offering concise, intuitive syntax that aligns with modern programming practices.

2 Proposal

1. `std::contains`: Check for a single value in a range.
2. `std::contains_any`: Verify if any specified values exist in a range.
3. `std::contains_all`: Ensure all specified values are present in a range.

3 Technical Specifications

- Defined for all STL containers, strings, and ranges.
- Utilizes concepts for compile-time type checks, ensuring compatibility and usability.
- Employs `std::ranges::find`, `std::ranges::any_of`, and `std::ranges::all_of` for implementation, guaranteeing efficiency.

4 Implementation

The following C++ code showcases the implementation of the proposed functions within the namespace `my_namespace`:

```cpp
#include <algorithm>
#include <concepts>
#include <ranges>

namespace my_namespace {

template <typename Range, typename T>
concept SearchableRange = requires(Range range, T value) {
    { std::ranges::find(range, value) } -> std::convertible_to<
    typename Range::iterator>;
};

template <std::ranges::input_range Range, typename T>
requires SearchableRange<Range, T>
constexpr bool contains(const Range& range, const T& value) {
    auto first = std::ranges::begin(range);
    auto last = std::ranges::end(range);
    return std::ranges::find(first, last, value) != last;
}

template <std::ranges::input_range Range, std::ranges::input_range
    Values>
requires SearchableRange<Range, typename Values::value_type>
constexpr bool contains_any(const Range& range, const Values&
    values) {
    return std::ranges::any_of(values, [&range](const auto& value)
    {
        return contains(range, value);
    });
}

template <std::ranges::input_range Range, std::ranges::input_range
    Values>
requires SearchableRange<Range, typename Values::value_type>
constexpr bool contains_all(const Range& range, const Values&
    values) {
    return std::ranges::all_of(values, [&range](const auto& value)
    {
        return contains(range, value);
    });
}

} // namespace my_namespace
```

Listing 1: Implementation of std::contains and Variants

5 Examples

5.1 contains all

```cpp
std::vector<std::string> systemsStatus = {"CalorimetersReady", "
    TrackingSystemsReady", "MagnetSystemsReady", "
    BeamInjectionSystemReady"};
bool allSystemsReady = std::contains_all(systemsStatus, {"
    CalorimetersReady", "TrackingSystemsReady", "MagnetSystemsReady
    ", "BeamInjectionSystemReady"});
```

Listing 2: System Readiness Check

5.2 contains all

```cpp
std::vector<double> magneticFieldStrengths = {3.8, 3.9, 4.0, 3.7};
bool fieldsWithinSpec = std::contains_all(magneticFieldStrengths,
    specificationLimits);
```

Listing 3: Magnetic Field Strength Verification

5.3 contains

```cpp
std::set<std::string> detectorReadings = {"NoPlasma", "BeamStable",
     "NoAnomalies"};
bool plasmaAbsent = std::contains(detectorReadings, "NoPlasma");
```

Listing 4: Plasma Presence Check

5.4 contains any

```cpp
std::vector<double> availableParticlesTeV = {13.6, 5.0, 7.5};
bool desiredParticleAvailable = std::contains_any(
    availableParticlesTeV, {13.6, 14.0});
```

Listing 5: Particle Energy Availability

These examples demonstrate the adaptability and power of the proposed functions, showcasing their potential to streamline complex checks and validations in the context.

6 Impact

- Enhances readability and maintainability of code.
- Reduces boilerplate, focusing on expressiveness and intent.
- Streamlines common container operations, aligning C++ with functionality present in other major languages.

7 Conclusion

The introduction of `std::contains`, `std::contains_any`, and `std::contains_all` represents a significant step towards modernizing C++ container manipulation, providing developers with powerful tools for writing clear, efficient, and robust code.