std::from_chars should work with std::string_view

Document #: D2007R0
Date: 2020-01-10
Project: Programming Language C++
Library Evolution Working Group Incubator
Library Evolution Working Group
Reply-to: Mateusz Pusz (Epam Systems)
<mateusz.pusz@gmail.com>

Contents

1 Introduction

`std::from_chars` accepts only a pair of raw `char` pointers in its overloads to provide a range of characters for conversion. The author proposes to add overloads taking `std::string_view` which is the main C++ vocabulary type to denote a range of characters.

2 Motivation and Scope

2.1 A naive approach

Unfortunately, it is common for an average C++ developer to provide the following implementation for converting a `std::string_view` to `int` via a `std::from_chars`.

```
void foo(std::string_view txt)
{
  int result;
  auto [ptr, ec] = std::from_chars(txt.begin(), txt.end(), result);
  if (ec == std::errc())
    // ...
}
```

The above will work for some implementations and build modes but will fail for others. This is caused by the fact that even though some vendors implement `std::string_view::const_iterator` in terms of `const char*`, such code is not portable. The C++ standard specifies `std::string_view::const_iterator` as *implementation defined* and only requires that the iterator's `value_type` is `char`.

2.2 A hacker's approach

After realizing the portability issue programmers often fix it in the following way which is far from what we want to teach:

```cpp
void foo(std::string_view txt)
{
  int result;
  auto [ptr, ec] = std::from_chars(&*txt.begin(), &*txt.end(), result);
  if (ec == std::errc())
    // ...
}
```

2.3 A correct approach

A correct approach works on the underlying data rather than on iterators which is counter intuitive as for C++ we always taught to use iterators to denote ranges:

```cpp
void foo(std::string_view txt)
{
  int result;
  auto [ptr, ec] = std::from_chars(txt.data(), txt.data() + txt.size(), result);
  if (ec == std::errc())
    // ...
}
```

3 Before/After Comparison

Before

```cpp
void foo(std::string_view txt)
{
  int result;
  auto [ptr, ec] = std::from_chars(txt.data()
  if (ec == std::errc())
    // ...
}
```

After

```cpp
void foo(std::string_view txt)
{
  int result;
  auto [ptr, ec] = std::from_chars(txt, result);
  if (ec == std::errc())
    // ...
}
```

4 Proposed Wording

The proposed changes are relative to the working draft of the standard as of [N4830].

Add the following to the end of synopsis in **20.19.1 [charconv.syn]**:

```cpp
from_chars_result from_chars(string_view txt, _see below_& value, int base = 10);
from_chars_result from_chars(string_view txt, float& value, chars_format fmt = chars_format::general);
from_chars_result from_chars(string_view txt, double& value, chars_format fmt = chars_format::general);
from_chars_result from_chars(string_view txt, long double& value, chars_format fmt = chars_format::genera
```

Update the following paragraphs in **20.19.3** [**charconv.from.chars**]:

1 All functions named `from_chars` analyze the provided string ~~`[first, last)`~~ for a pattern, where string is either a `txt` or `[first, last)` denoting ~~`[first, last)` is required to be~~ a valid range.

```
  from_chars_result from_chars(const char* first, const char* last,
                               _see below_& value, int base = 10);
+ from_chars_result from_chars(string_view txt,
+                              _see below_& value, int base = 10);
```

```
  from_chars_result from_chars(const char* first, const char* last, float& value,
                               chars_format fmt = chars_format::general);
+ from_chars_result from_chars(string_view txt, float& value,
+                              chars_format fmt = chars_format::general);
  from_chars_result from_chars(const char* first, const char* last, double& value,
                               chars_format fmt = chars_format::general);
+ from_chars_result from_chars(string_view txt, double& value,
+                              chars_format fmt = chars_format::general);
  from_chars_result from_chars(const char* first, const char* last, long double& value,
                               chars_format fmt = chars_format::general);
+ from_chars_result from_chars(string_view txt, long double& value,
+                              chars_format fmt = chars_format::general);
```

5 Acknowledgements

Special thanks and recognition goes to Epam Systems for supporting my membership in the ISO C++ Committee and the production of this proposal.

6 References

[N4830] Richard Smith. 2019. Committee Draft, Standard for Programming Language C++.
https://wg21.link/n4830