Item 3: Case-Insensitive Strings, Part 2

Le Van Huynh

October 22, 2023

Contents

1 Questions

Consider the solution in Item 2.

1. Is it safe to inherit `ci_char_traits` from `char_traits<char>` this way?
2. Why does the following code fail to compile?

```
ci_string s = "abc";
cout << s << endl;
```

3. What about using other operators (for example, `+`, `+=`, `=`) and mixing strings and `ci_strings` as arguments? For example:

```
string    a = "aaa";
ci_string b = "bbb";
string    c = a + b;
```

2 Answers

1. Yes, because `ci_char_traits` is never used polymorphically through a pointer or reference to the base class `char_traits`.

2. The standard provides operator `<<` with the following signature:

```
```

The operator only support `basic_ostream` and `basic_string` with the same `Traits` parameter. However, `ci_string` and `cout` have different `Traits` parameter:

1. `cout` is of type `ostream` and has the `Traits` parameter set to `std::char_traits`:

```
template<
    class CharT,
    class Traits = std::char_traits<CharT>
> class basic_ostream;
```

2. `ci_string` as we implemented, use `Traits = ci_char_traits`:

```
using ci_string = std::basic_string<char, ci_char_traits>;
```

Therefore, `cout` doesn't work with `ci_string`.

1. We need to decide the meaning of these operators, then we can define them ourselves or use `.c_str()` to use the existing operators.