C++ constexpr parlor tricks: How can I obtain the length of a string at compile time?

devblogs.microsoft.com/oldnewthing/20221114-00

November 14, 2022

Raymond Chen

Say you want to obtain, at compile time, the length of a compile-time string constant. The problem is that the `strlen` function is not `constexpr`.

```
#include <cstring>

template<std::size_t> void tryme();

void test()
{
    // error; expression did not evaluate to a constant
    char buffer[strlen("hello") + 1];

    // error: invalid template argument
    tryme<strlen("hello")>();

    switch (0)
    {
    // error: case expression is not constant
    case strlen("hello"): break;
    }
}
```

Note that gcc and clang support variable-length arrays as a nonstandard extension, so you may get away with the `buffer` declaration unless you turn off that extension. In fact, gcc goes further and accepts all three statements!

How can you get all three of the above to work in standard-conforming code? One idea is to write your own `constexpr_strlen`. But it turns out that somebody already wrote it for you, although it has a rather awkward name: `std::char_traits<T>::length()`.

```
#include <string>

constexpr std::size_t constexpr_strlen(const char* s)
{
    return std::char_traits<char>::length(s);
    // or
    return std::string::traits_type::length(s);
}

constexpr std::size_t constexpr_wcslen(const wchar_t* s)
{
    return std::char_traits<wchar_t>::length(s);
    // or
    return std::wstring::traits_type::length(s);
}
```

Raymond Chen

Follow