Reconsidering literal operator templates for strings

Document #:	D0424R1
Date:	2017-10-12
Project:	Programming Language C++
Audience:	Evolution Working Group

Reply-to:	Louis Dionne <ldionne.2@gmail.com>
	Hana Dusíková <hanicka@hanicka.net>

1 Revision history

- R0 – Initial draft
- R1 – Rewrite with different UDL form per EWG direction, and update motivation

2 Introduction

C++11 added the ability for users to define their own literals suffixes. Several forms of literal operators are available, but none of them allows getting a string literal as a compile-time entity from within the user-defined literal operator. This prevents the user-defined literal to create an object whose type depends on the *contents* of the string literal. This paper proposes solving that problem by allowing user-defined literal operators of the following form to be considered for string literals:

```
template <typename CharT, CharT const* str, std::size_t length>
auto operator"" _udl();

auto x = "abcd"_udl; // calls the above function
```

3 History

1. [N3599] (in 2013) proposed adding the missing literal operator using a `char...` parameter pack, but the paper was rejected at that time with the following conclusion ([CWG66]):

   > Revise with additional machinery for compile time string processing

2. [P0424R0] (the initial revision of this paper) was presented in Issaquah in 2016 and argued for adding the missing literal operator using a `char...` parameter pack, but the paper was rejected because implementers were concerned with the compile-time cost of instantiating a function template with such a parameter pack.

3. This paper addresses implementer's concerns by using a `char const*` template parameter instead of a `char...` parameter pack.

4 Motivation

Many use cases have recently come up, the most notable ones being compile-time JSON parsing and compile-time regular expression parsing. For example, a regular expression engine can be generated at compile-time as follows:

```
#include "pregexp.hpp"
using namespace sre;

auto regexp = "^(?:[abc]|xyz).+$"_pre;

int main(int argc, char** argv) {
  if (regexp.match(argv[1])) {
    std::cout << "match!" << std::endl;
    return EXIT_SUCCESS;
  } else {
    std::cout << "no match!" << std::endl;
    return EXIT_FAILURE;
  }
}
```

Under the hood of [CTRE] library, constexpr functions and template metaprogramming are used to parse the string literal and generate a type like the following from the string literal:

```
RegExp<
  Begin,
  Select<Char<'a','b','c'>, String<'x','y','z'>>,
  Plus<Anything>,
  End
>
```

Since the regular expression parser is generated at compile-time, it can be better optimized and the resulting code is much faster than `std::regex` (3000x faster has been witnessed).

Similar functionality has traditionally been achieved by using expression templates and template metaprogramming to build the representation of the regular expression instead of simply parsing the string at compile-time. For example, the same regular expression with [Boost.Xpressive] looks like this:

```
auto regexp = bos >> ((set='a','b','c')|(as_xpr('x') >> 'y' >> 'z')) >> +_ >> eos;
```

It is worth noting that the specific use case of parsing regular expressions at compile-time came up at CppCon during a lightning talk, and the room showed a very strong interest in getting a standardized solution to this problem. Today, we must rely on a non-standard extension provided by Clang and GCC.

5 How would that work?

The idea behind how this operator would work is that the compiler would generate a constexpr string and pass that to the user-defined literal. For example:

```
template <typename CharT, CharT const* str, std::size_t length>
auto operator"" _udl();

"foobar"_udl;

// should be roughly equivalent to

constexpr char __unnamed[] = "foobar";
operator""_udl<char, __unnamed, sizeof(__unnamed)-1>();
```

Calling a function template with such a template-parameter-list works in Clang and GCC today.

6 Implementation experience

A very similar literal operator is already provided by both Clang and GCC:

```
template <typename CharT, CharT ...s>
constexpr auto operator"" _udl(); // works
```

It is expected that implementing support for the proposed user-defined literal operator should be easy.

7 Wording

Wording will be provided if the proposal makes it through EWG.

8 References

[N3599] Richard Smith, *Literal operator templates for strings*
http://open-std.org/JTC1/SC22/WG21/docs/papers/2013/n3599.html

[P0424R0] Louis Dionne, *Reconsidering literal operator templates for strings*
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/p0424r0.pdf

[CWG66] Richard Smith, *EWG Issue #66*
http://cplusplus.github.io/EWG/ewg-active.html#66

[Boost.Xpressive] Eric Niebler, *Boost.Xpressive*
http://www.boost.org/doc/libs/release/doc/html/xpressive.html

[CTRE] Hana Dusíková *Compile Time Regular Expression library*
https://github.com/hanickadot/compile-time-regular-expressions