String literals have the wrong type

Jonathan Müller — @foonathan

Ranges and string literals

o73ax86v4

```
std::cout << std::ranges::size("abc") << '\n';
```

think-cell

Ranges and string literals

o73ax86v4

```
std::cout << std::ranges::size("abc") << '\n';
```

```
4
```

think-cell

String literal type

The type of string literal is array of `const char`.

This includes the null terminator.

think-cell

Ranges and string literals

azrMj31Wh

```
for (auto c : "abc")
    std::cout << int(c) << '\n';
```

```
97
98
99
0
```

think-cell

std::begin/std::end

```
namespace std
{
    template <typename T, std::size_t N>
    T* begin(T (&array)[N])
    {
        return array;
    }

    template <typename T, std::size_t N>
    T* end(T (&array)[N])
    {
        return array + N;
    }
}
```

think-cell

Guideline

Never use string literals with ranges!

They have the wrong type.

think-cell

Philosophy: Strings are just ranges.

Philosophy: Strings are just ranges.

```
fmt::format("The answer is {}.\n", 42);
```

Philosophy: Strings are just ranges.

```
fmt::format("The answer is {}.\n", 42);
```

```
tc::concat("The answer is ", tc::as_dec(42), ".\n");
```

Workaround

```cpp
namespace tc
{
    template <typename T, std::size_t N>
        requires tc::char_type<T>
    T* begin(T (&array)[N])
    {
        return array;
    }

    template <typename T, std::size_t N>
        requires tc::char_type<T>
    T* end(T (&array)[N])
    {
        return array + N - 1;
    }
}
```

UDLs to the rescue

```
namespace tc {
    struct string_literal {
        const char* ptr;
        size_t length;

        const char* begin() const  { return ptr; }
        const char* end() const { return ptr + length; }
        operator const char*() const { return ptr; }
    };
}

tc::string_literal operator""_tc(const char* ptr, std::size_t length) {
    return {ptr, length};
}
```

think-cell

Ranges and UDLs

```
std::cout << std::ranges::size("abc"_tc) << '\n';
for (auto c : "abc"_tc)
    std::cout << int(c) << '\n';
```

```
3
97
98
99
```

think-cell

Ranges and UDLs

```
std::cout << std::ranges::size("abc"_tc) << '\n';
for (auto c : "abc"_tc)
    std::cout << int(c) << '\n';
```

```
3
97
98
99
```

```
tc::concat("The answer is "_tc, tc::as_dec(42), ".\n"_tc);
```

think-cell

C APIs and string literals

```
HANDLE open(const char* path); // some C API
```

```
auto file = open("foo.bar");
```

think-cell

C APIs and string literals

```
HANDLE open(const char* path); // some C API
```

```
constexpr auto path_dir = "/some/path";
…
auto file = open((std::string(path_dir) + "foo.bar").c_str());
```

think-cell

C APIs and string literals

```
HANDLE open(const char* path); // some C API
```

```
#define PATH_DIR "/some/path"
…
auto file = open(PATH_DIR "foo.bar");
```

think-cell

A literal range type

```
namespace tc
{
    template <typename T, auto ... Ts>
    struct literal_range
    {
        static constexpr T array[] = {T(Ts)..., T(0)};

        const T* begin() { return array; }
        const T* end() { return array + sizeof...(Ts); }

        operator const T*() const { return array; }
    };
}
```

think-cell

UDLs to the rescue (2)

```
template <tc::string_template_param Str>
auto operator""_tc()
{
    return []<std::size_t ... Idx>(std::index_sequence<Idx...>) {
        using char_type = typename decltype(Str)::char_type;
        return tc::literal_range<char_type, Str[Idx]...>;
    }(std::make_index_sequence<Str.size()>{});
}
```

think-cell

UDLs to the rescue (2)

```
template <tc::string_template_param Str>
auto operator""_tc()
{
    return []<std::size_t ... Idx>(std::index_sequence<Idx...>) {
        using char_type = typename decltype(Str)::char_type;
        return tc::literal_range<char_type, Str[Idx]...>;
    }(std::make_index_sequence<Str.size()>{});
}
```

```
"abc"_tc // tc::literal_range<char, 'a', 'b', 'c'>
```

think-cell

Literal range concatenation

```
namespace tc
{
    template <typename T, auto ... Ts, typename U, auto ... Us>
      requires tc::has_common_type<T, U>
    auto concat(literal_range<T, Ts...>, literal_range<U, Us...>)
    {
        return literal_range<std::common_type_t<T, U>, Ts..., Us...>{};
    }
}
```

think-cell

C APIs and UDLs

```
HANDLE open(const char* path); // some C API
```

```
constexpr auto path_dir = "/some/path"_tc;
…
auto file = open(tc::concat(path_dir, "foo.bar"_tc));
```

think-cell

Bonus: Correct character type

ASCII `"abc"_tc` → `tc::literal_range<tc::char_ascii, …>`

think-cell

Bonus: Correct character type

ASCII `"abc"_tc` → `tc::literal_range<tc::char_ascii, …>`

UTF-8 `u8"abc"_tc` → `tc::literal_range<char, …>`

think-cell

Bonus: Correct character type

ASCII `"abc"_tc` → `tc::literal_range<tc::char_ascii, …>`

UTF-8 `u8"abc"_tc` → `tc::literal_range<char, …>`

UTF-16 `u"abc"_tc` → `tc::literal_range<wchar_t, …>` on Windows

think-cell

Bonus: Cheap transcoding

```
template <typename CharT>
auto convert_enc(auto&& rng)
{
    // return a rng transcoded to CharT
}
```

think-cell

Bonus: Cheap transcoding

```cpp
template <typename CharT>
auto convert_enc(auto&& rng)
{
    // return a rng transcoded to CharT
}
```

```cpp
template <typename CharT, auto ... Cs>
auto convert_enc(tc::literal_range<tc::char_ascii, Cs...>)
{
    return tc::literal_range<CharT, Cs...>{};
}
```

think-cell

Conclusion

github.com/think-cell/think-cell-library

We're hiring: think-cell.com/cpponsea

@foonathan@fosstodon.org
youtube.com/@foonathan

think-cell