CppIndia

String View

by Mahendra Garodi

Agenda

Problems with std::string

```
std::string extractMessage(std::string line)
{
    return line.substr(5);
}

std::string myMessage{"INFO This is information"};
std::cout << extractMessage(myMessage); //Call 1

std::cout << extractMessage("WARN This is warning"); //Call 2
```

Problems with const std::string&

```
std::string extractMessage(const std::string& line)
{
    return line.substr(5);
}

std::string myMessage{"INFO This is information"};
std::cout << extractMessage(myMessage); //Call 1

std::cout << extractMessage("WARN This is warning"); //Call 2
```

What's happening

- Temporary Strings
- Allocation on heap
  - Sometime SSO might help here
- Copying of data

What is string_view?

- The class template basic_string_view describes an object that can refer to a constant contiguous sequence of char-like objects
- non-owning read-only view into a subsequence of a string, implementable as either a pair of pointers or a pointer and a length
- Introduced in C++17 as part of standard library

Constructors

- `constexpr basic_string_view() noexcept;`
- `constexpr basic_string_view(const basic_string_view& other) noexcept = default;`
- `constexpr basic_string_view(const CharT* s, size_type count);`
- `constexpr basic_string_view(const CharT* s);`
- ```
  template<class It, class End>
  constexpr basic_string_view(It first, End last);
  ```
  ```

Interface

- Almost same as `const std::string&`
  - Accessors - `at(), front(), back(), data(), operator[]`
  - Iterators - `begin()-end(), rbegin()-rend(), cbegin()-cend(), crbegin()- crend()`
  - Capacity - `size()/length(), max_size(), empty()`
  - Operations - `copy(), substr(), comparisons`
- Additional functionality like
  - `remove_prefix(), remove_suffix()`

remove_prefix & remove_suffix

```
std::string message = "This is very very long string";
std::string_view message_view = message;
std::cout << message_view << '\n';

message_view.remove_prefix(5);
message_view.remove_suffix(7);
std::cout << message_view;
```

Output:

```
This is very very long string
is very very long
```

Conversions

Source	Destination	Using
const char*	std::string	Non-explicit constructor
const char*	std::string_view	Non-explicit constructor
std::string	std::string_view	Conversion Operator
std::string_view	std::string	Explicit Constructor

➔ Why *string_view* -> *string* conversion is explicit?

Assignment & Compare semantics

```
auto s1 = "This is tricky"sv;
auto s2 = s1;
auto s3 = "This is tricky"sv;

auto res1 = s1 == s2;
auto res2 = s2 == s3;
std::cout << std::boolalpha << res1 << ", " << res2;
```

- ➔ Assignment has shallow semantics
- ➔ Comparison has deep semantics

const std::string& Vs std::string_view

```
std::string getString()
{
 return std::string("A very very long string");
}

//Call 1
const std::string& str = getString();
std::cout << str << '\n';

//Call 2
std::string_view str_view = getString();
std::cout << str_view << '\n';
```

const std::string& Vs std::string_view

```
std::string message = "This is very very long string";
const std::string& msg = message;
std::string_view msg_view = message;

message = "What do you think?";

std::cout << msg << '\n';


std::cout << msg_view << '\n';
```

Undefined Behavior

Containers & String View

```
std::vector<std::string_view> elements;

void save(const std::string& elem)
{
    elements.push_back(elem);
}
```

Implicit Conversion

- ➔ Storing string_view in container is potentially risky
- ➔ You can end up holding onto freed memory (temporary strings)

Containers & String View

```
std::map<std::string, int> frequencies;

int getFrequency(std::string_view keyword)
{
    if ( auto it = frequencies.find(keyword); it != frequencies.end())
    {
        return it->second;
    }

    return 0;
}
```

Compilation Error

Containers & String View

```
std::map<std::string, int, std::less<>> frequencies;

int getFrequency(std::string_view keyword)
{
    if ( auto it = frequencies.find(keyword); it != frequencies.end())
    {
        return it->second;
    }

    return 0;
}
```

String View Literal

```
// PART1
std::string_view str_view = "This is very long string"s;
std::cout << str_view;
```

Runtime Error

```
// PART2
std::string_view s2 = "abc\0\0def";
std::string_view s3 = "abc\0\0def"sv;
std::cout << s2.size() << ", " << s3.size();
```

Output: 3, 8

When to use string::view

- Passing as parameter to a function
- Returning from a function
- A reference to part of a long-lived data structure

Risks

- Taking care of the (non)null-terminated strings
  - Problematic when calling functions like atoi, printf that accepts null-terminated strings
  - Conversion into strings
- References and Temporary objects
  - When returning string_view from a function
  - Storing string_view in objects or container

Alternative to std::string_view

- absl::string_view
- StringPiece : Google
- StringRef : LLVM
- boost::string_ref
- folly::Range

Performance

Performance of std::string_view vs std::string from C++17

- Substring
- String Split with Iterators
- String Split with Raw Pointers

"The standard library saves programmers from having to reinvent the wheel"

- Bjarne Stroustrup