Handling Non-Null Terminated Strings

A string_view works well with null-terminated strings. Let's see what happens when we point it to a substring which does not have a null character at the end.

If you get a string_view from a string then it will point to a null-terminated chunk of memory:

```
#include <iostream>
#include <string>
using namespace std;

int main() {
    std::string s = "Hello World";
    std::cout << s.size() << '\n';
    std::string_view sv = s;
    std::cout << sv.size() << '\n';
}</pre>
```

The two cout statements will both print 11.

But what if you have just a part of the string:

```
#include <iostream>
using namespace std;

int main() {
   std::string s = "Hello World";
   std::cout << s.size() << '\n';
   std::string_view sv = s;
   auto sv2 = sv.substr(0, 5);
   std::cout << sv2.data() << '\n'; /// ooops?
}</pre>
```

sv2 should contain only "Hello", but when you access the pointer to the underlying memory, you'll receive the pointer to the whole string. The

expression: cout << sv2.data() will print the whole string, and not just a part

of it! sv2.data() returns the pointer to the "Hello World" character array inside the string s object.

Of course, when you print sv2 you'll get the correct output.

```
#include <iostream>
using namespace std;

int main() {
   std::string s = "Hello World";
   std::cout << s.size() << '\n';
   std::string_view sv = s;
   auto sv2 = sv.substr(0, 5);
   std::cout << sv2 << '\n'; /// ooops?
}</pre>
```

This is because std::cout handles string_view type.

The example shows a potential problem with all third-party APIs that assume null-terminated strings. We'll go through them one by one.